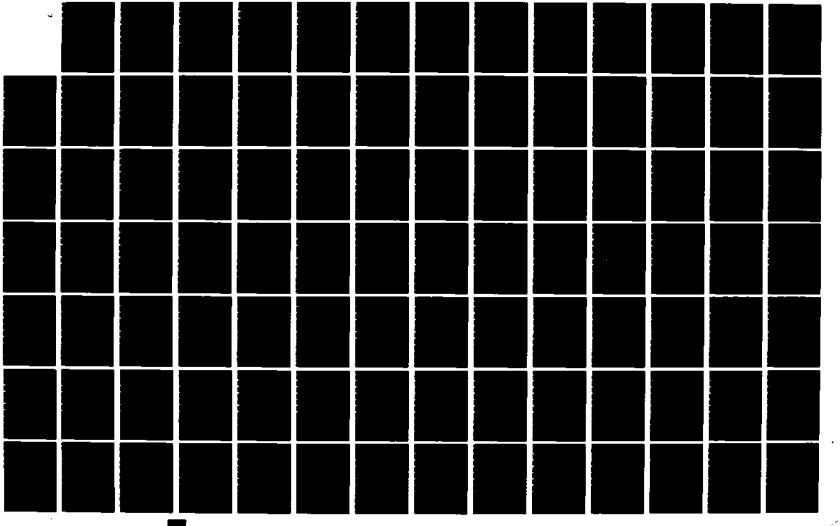
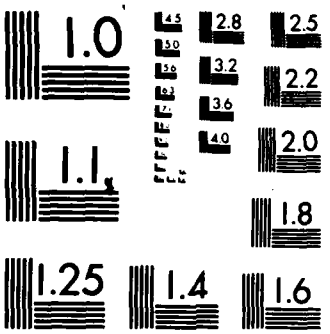
AD-A137 828 DISTRIBUTED KNOWLEDGE BASE SYSTEMS FOR DIAGNOSIS AND INFORMATION RETRIEVA..(U) OHIO STATE UNIV COLUMBUS DEPT OF COMPUTER AND INFORMATION SCI.. B CHANDRASEKARAN 1/2

UNCLASSIFIED NOV 83 AFOSR-TR-84-0039 AFOSR-82-0255 F/G 9/4 NL





AFOSR-TR- 84-0039

RF Project 763180/714659
Annual Report

DISTRIBUTED KNOWLEDGE BASE SYSTEMS FOR DIAGNOSIS AND INFORMATION RETRIEVAL

B. Chandrasekaran
Department of Computer and Information Science

For the Period
July 1, 1982 - July 30, 1983

DEPARTMENT OF THE AIR FORCE
Air Force Office of Scientific Research
Bolling Air Force Base, D.C. 20332

Grant No. AFOSR-82-0255

November, 1983

OSU
The Ohio State University
Research Foundation
1314 Kinnear Road
Columbus, Ohio 43212

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 763180/714659	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 84-0039
6a. NAME OF PERFORMING ORGANIZATION Ohio State University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code) Department of Computer and Information Science Columbus OH 43212	7b. ADDRESS (City, State and ZIP Code) Directorate of Mathematical and Information Sciences, Bolling AFB DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR 8b. OFFICE SYMBOL (If applicable) NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-82-0255
8c. ADDRESS (City, State and ZIP Code) Bolling AFB DC 20332	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A7; WORK UNIT NO.
11. TITLE (Include Security Classification) "Distributed Knowledge Base Systems For Diagnosis and Information Retrieval"	
12. PERSONAL AUTHOR(S) B. Chandrasekaran	
13a. TYPE OF REPORT Annual 13b. TIME COVERED FROM 7/1/82 TO 6/30/83	14. DATE OF REPORT (Yr., Mo., Day) NOV 83 15. PAGE COUNT 39
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Expert systems; types of problem solving; expert systems for design; functional representations; deep knowledge representations.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

During the year, progress was made in a number of directions: (1) A better understanding of different types of problem solving that underlie expert reasoning was obtained. (2) Advances in representing design knowledge as plans in design specialists were made. (3) CSRL, the language for diagnostic expert system building that was designed in the author's laboratory, was applied to the implementation of a diagnostic system for the fuel system of an automobile and directions for new constructs for the language were obtained. (4) A representation for functional understanding of how a device works was obtained, and methods of automatically generating diagnostic expert systems from this representation of a device were also obtained. (5) An analysis of how techniques and tasks can be matched in expert design was undertaken.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☒ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Joseph Bram	22b. TELEPHONE NUMBER (Include Area Code) (202) 767-4939 22c. OFFICE SYMBOL NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

Annual REPORT OF SCIENTIFIC PROGRESS

ON GRANT AFOSR 82-0255

FOR RESEARCH ON

"DISTRIBUTED KNOWLEDGE BASE SYSTEMS

FOR DIAGNOSIS AND INFORMATION RETRIEVAL"

FOR PERIOD 1 July, 1982 -- 30 June, 1983

B. Chandrasekaran, Principal Investigator
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43201

Submitted November 1983 to

Air Force Office of Scientific Research
Bolling Air Force base, DC 20332

Chief, Technical Information Division

Technical Progress

In the original proposal we had outlined a long term program for conducting research in knowledge based systems. In particular we proposed to study issues in diagnostic reasoning and in knowledge-directed information retrieval. During the first year most of the progress came in the area of diagnostic reasoning and in the conceptual foundations of knowledge-based systems in general. We also developed an approach to a new type of task: design of mechanical parts.

In particular, the following specific progress was made. (We will summarize the nature of the result here and attach a paper in each case that gives details of the results.)

1. We have elaborated our theory of types of problem solving that underlies expert reasoning. The idea is that a complex task can often be broken down into a number of generic tasks, for each of which a particular problem solving regime is appropriate. Each of these tasks can be solved by a collection of conceptual specialists among whom knowledge of the domain is distributed. These specialists solve the problem by engaging in that generic type of problem solving by exchanging messages of specific types. We have enclosed as appendix a paper, "Towards a Taxonomy of Problem Solving Types," which appeared in the AI Magazine, which gives the details of the theory.

2. In earlier work we had developed approaches to three generic types of problem solving: diagnosis, knowledge-directed data retrieval, What-Will-Happen-If type of reasoning. During the period of research under report, we formulated another important type of problem solving: design by refining plans. We have been applying the approach to the implementation of an expert system for mechanical design. The attached paper, "An Approach to Expert Systems for Mechanical Design," was presented at the IEEE Computer Society, Trends & Applications Conference.

3. We have developed (with support from another source) a tool for efficient construction of diagnostic expert systems. This tool is a high level language called CSRL. Under this grant support we have been experimenting with the application of this tool to the design and implementation of expert systems in the area of mechanical systems, since that was one of the domains that we emphasized in the original proposal. We reported on this language

at the last International Joint Conference on Artificial Intelligence at Karlsruhe. The paper from that Proceedings reporting on CSRL is enclosed. We also include with this report another technical report that discusses our experience in using this tool in the construction of an expert system for fuel systems for automobiles.

4. We have been investigating the issues related to how an expert system may have "deep" knowledge of its domain and use it to do problem solving, as opposed to the current generation of expert systems that use what one might call "compiled" knowledge. E.g. all the current expert systems in medicine have knowledge relating symptoms/manifestations and diseases explicitly encoded in the knowledge base. However, often a person who has an understanding of the domain will be able to derive these relationships from a deeper model. We have developed a language in which the understanding of an agent about how a device works may be encoded. This language expresses how a function of a device may be related to the behavior and structure of it and its components. In addition we have developed a compiler which can work on this functional representation and produce a diagnostic expert system. This result is of considerable significance we think, since it will enable for the first time a representation of "understanding" of a device. We have applied this methodology to the representation of the functions of a household electric buzzer and show how the compiler generates a dignostic problem solver from this. A paper reporting on this is attached as appendix to the report.

5. We have been looking into how capabilities of various expert system approaches can be characterized. A methodology by which a complex real world decision task may be decomposed into generic tasks and techniques suited for various generic tasks can then be applied is outlined in another attached paper, "Expert Systems: Matching Techniques to Tasks," which was presented as an invited talk at the New York University Symposium on Expert Systems for Business Applications. This will shortly appear as an article in a book of that title.

Personnel Activities

Two items of interest need to be mentioned here. Prof. B. Chandrasekaran, the PI for the Grant, spent 3 months at the MIT Laboratory for Computer Science as a Visiting Scientist during the research period. He worked with Prof. Peter Szolovits and Prof. Ramesh Patil on several aspects of expert systems. He also spent one month at Carnegie Mellon University under the sponsorship of Prof. A. Newell. A portion of his support for these visits came from the

AFOSR Grant. In addition to these major visits, Prof. Chandrasekaran gave a number of talks at BBN, GTE Labs, NRL AI Lab., and other places over the year.

Mr. Tom Bylander, a Graduate Research Associate under the Grant, won an award for travel to the International Joint Conference on Artificial Intelligence to present the paper on CSRL.

Computing Environment

Quite a bit of our effort went into gearing up for the introduction of Lisp machines into our computing environment. These machines will be arriving shortly. A number of changes will need to be made in the language environment: we are moving into an Interlisp environment, and many of our tools are being recoded for that environment.

LIST OF PAPERS IN APPENDIX

1. "Towards a Taxonomy of Problem Solving Types" by B. Chandrasekaran

2. "An Approach to Expert Systems for Mechanical Design" by David C. Brown and B. Chandrasekaran

3. "CSRL: A Language for Expert Systems for Diagnosis" by Tom Bylander, Sanjay Mittal, and B. Chandrasekaran

4. "Application of the CSRL Language to the Design of Expert Diagnosis Systems: The Auto-Mech Experience" by Michael C. Tanner and Tom Bylander

5. "A Representation for the Functioning of Devices that Supports Compilation of Expert Problem Solving Structures" by V.S. Moorthy and B. Chandrasekaran

6. "Expert Systems: Matching Techniques to Tasks" by B. Chandrasekaran

Towards a Taxonomy Of Problem Solving Types

B. Chandrasekaran

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210 USA

Abstract

Our group's work in medical decision making has led us to formulate a framework for expert system design, in particular about how the domain knowledge may be decomposed into substructures. We propose that there exist different problem-solving types, i.e., uses of knowledge, and corresponding to each is a separate substructure specializing in that type of problem-solving. Each substructure is in turn further decomposed into a hierarchy of specialists which differ from each other not in the type of problem-solving, but in the conceptual content of their knowledge: e.g., one of them may specialize in "heart disease," while another may do so in "liver," though both of them are doing the same type of problem-solving. Thus ultimately all the knowledge in the system is distributed among problem-solvers which know how to use that knowledge. This is in contrast to the currently dominant expert system paradigm which proposes a common knowledge base accessed by knowledge-free problem-solvers of various kinds. In our framework there is no distinction between knowledge bases and problem-solvers: each knowledge source *is* a problem-solver. We have so far had occasion to deal with three generic problem-solving types in expert clinical reasoning: diagnosis (classification), data retrieval and organization, and reasoning about consequences of actions. In a novice, these expert structures are often incomplete, and other knowledge structures and learning processes are needed to construct and complete them.

This is a revised and extended version of an invited talk entitled, "Decomposition of Domain Knowledge Into Knowledge Sources: The MDX Approach," delivered at the IV National Conference of the Canadian Society for Computational Studies of Intelligence, May 17-19, 1982, Saskatchewan.

Introduction

For the past few years our research group has been investigating the issues of problem-solving as well as knowledge organization and representation in medical decision making. In parallel with this investigation we have also been building and extending a cluster of systems for various aspects of medical reasoning. The major system in this cluster is MDX, which is a diagnostic system, i.e., its role is to arrive at a classification of a given case into a node of a diagnostic hierarchy. The theoretical basis of this diagnostic problem-solving is laid out in some detail in Gomez and Chandrasekaran.

The MDX system, which is wholly diagnostic in its knowledge, communicates with two auxiliary systems, PATREC and RADEX. PATREC is a data base assistant in the sense it acquires the patient data, organizes them, and answers the queries of MDX concerning the patient data. In all these activities PATREC uses various types of inferential knowledge embedded in an underlying conceptual model of the domain of medical data. RADEX is a radiology consultant to MDX, and it suggests or confirms diagnostic possibilities by reasoning based on its knowledge of imaging procedures and relevant anatomy. See Mittal and Chandrasekaran (Mittal, Chandrasekaran, 1981) and Chandrasekaran et al (Chandrasekaran, Mittal and Smith, 1980) for further details about these subsystems.

Though in a sense RADEX and PATREC can both be viewed as "intelligent" data base specialists, RADEX has some additional features of interest due to the perceptual nature of some of its knowledge. However, for the purpose of this paper, it is not necessary to go into RADEX in much detail, and we can view PATREC as prototypical of this class of auxiliary systems.

Our aim in this paper is to outline a point of view about how a domain gets naturally decomposed into substructures each of which specializes in one type of problem-solving. Each of these substructures in turn further gets decomposed into small knowledge sources of the same problem-solving type, but specializing in different concepts in the domain. We shall see that this sort of decomposition results in more natural control and focus properties of the overall system. Identification of these substructures and how they communicate with one another is vital to the proper *organization* of the body of knowledge for problem-solving in that domain.

Our method in this paper will be to examine how knowledge is used in a few well-defined tasks: diagnosis, data storage and retrieval, and reasoning about consequences of actions. It should be emphasized that these tasks are not particular to the medical domain. Rather they are fundamental generic tasks occurring in a wide variety of problem-solving situations. Thus these tasks are elements of a taxonomy of basic problem-solving types. When we are done with this examination, the general principles of knowledge decomposition will begin to take on some clarity.

One final point: we will use examples from both medical and non-medical domains. In particular, there are many similarities between reasoning about diseases and therapies on one hand and trouble-shooting and synthesis of corrective actions in complex engineering systems on the other.

The Diagnostic Task

By the term "diagnostic task," we mean something very specific: the identification of a case description with a specific node in a pre-determined diagnostic hierarchy. For the purpose of current discussion let us assume that all the data that can be obtained are already there, i.e., the additional problem of launching exploratory procedures such as ordering new tests etc. does not exist. The following brief account is a summary of the more detailed account given in (Gomez, Chandrasekaran, 1981) of diagnostic problem-solving.

Let us imagine that corresponding to each node of the classification hierarchy alluded to earlier we identify a "concept." The total diagnostic knowledge is then distributed through the conceptual nodes of the hierarchy in a specific manner to be discussed shortly. The problem-solving for this task will be performed top down, i.e., the top-most concept will first get control of the case, then control will pass to an appropriate successor concept, and so on. In the medical example, a fragment of such a hierarchy might be as shown in Fig. 1.

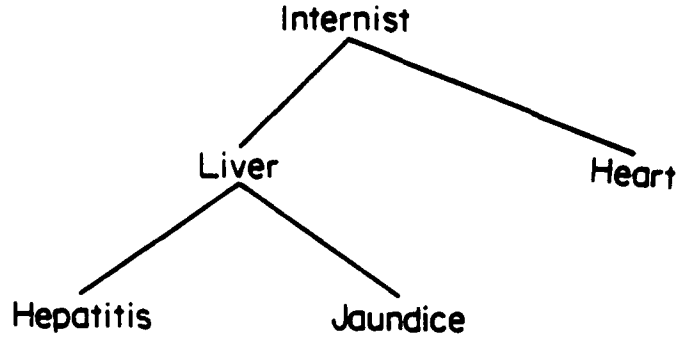


Figure 1.

More general classificatory concepts are higher in the structure, while more particular ones are lower in the hierarchy. It is as if INTERNIST first establishes that there is in fact a disease, then LIVER establishes that the case at hand is a liver disease, while say HEART etc. reject the case as being not in their domain. After this level, JAUNDICE may establish itself and so on.

Each of the concepts in the classification hierarchy has "how-to" knowledge in it in the form of a collection of *diagnostic rules*. These rules are of the form: <symptoms> → <concept in hierarchy>, e.g., "If high SGOT, add n units of evidence in favor of cholestasis." Because of the fact that when a concept rules itself out from relevance to a case, all its successors also get ruled out, large portions of the diagnostic knowledge structure never get exercised. On the other hand, when a concept is properly invoked, a small, highly relevant set of rules comes into play.

The problem-solving that goes on in such a structure is *distributed*. The problem-solving regime that is implicit in the structure can be characterized as an *"establish-refine"* type. That is, each concept first tries to establish or reject itself. If it succeeds in establishing itself, the refinement process consists of seeing which of *its* successors can establish itself. Each concept has several clusters of rules: confirmatory rules, exclusionary rules, and perhaps some recommendation rules. The evidence for confirmation and exclusion can be suitably weighted and combined to arrive at a conclusion to establish, reject or suspend it. The last mentioned situation may arise if there is not sufficient data to make a decision. Recommendation rules are further optimization devices to reduce the work of the subconcepts. Further discussion of this type of rules is not necessary for our current purpose.

The concepts in the hierarchy are clearly not a static collection of knowledge. They are active in problem-solving. They also have knowledge only about establishing or rejecting the relevance of that conceptual entity. Thus they may be termed "specialists," in particular, "diagnostic specialists." The entire collection of specialists engages in distributed problem-solving.

The above account of diagnostic problem-solving is quite incomplete. We have not indicated how multiple diseases can be handled within the framework above, in particular

when a patient has a disease secondary to another disease. Gomez has developed a theory of diagnostic problem-solving which enables the specialists in the diagnostic hierarchy to communicate the results of their analysis to each other by means of a *blackboard* (Erman, Lesser, 1975), and how the problem-solving by different specialists can be coordinated. See (Gomez, Chandrasekaran, 1981) for details. Similarly, how the specialists combine the uncertainties of medical data and diagnostic knowledge to arrive at a relatively robust conclusion about establishing or rejecting a concept is an important issue, for a discussion of which we refer the reader to (Chandrasekaran, Mittal and Smith, 1982).

The points to notice here are the following. The control transfer from specialist to specialist is akin to the corresponding situation in the medical community. We shall have more to say about this later on. Especially note that there is no "problem-solver" standing outside, *using* a knowledge base. The hierarchy of diagnostic specialists is the problem-solver as well as the knowledge-base, albeit of a limited type and scope. That is, the particular kind of problem-solving is *embedded* in each of the units in the knowledge structure.

Data Retrieval and Inference

Consider the following situation that might arise in diagnostic problem-solving that was discussed earlier. Suppose a rule in the liver specialist was: "If history of anesthetic exposure, consider hepatitis." This is a legitimate diagnostic rule in the sense described earlier, i.e., it relates a manifestation to a conceptual specialist. However, suppose there is no mention of anesthetics in the patient record, but his history indicates recent major surgery. We would expect a competent physician to infer possible exposure to anesthetics in this case and proceed to consider hepatitis. Similarly, if a diagnostic rule has "abdominal surgery" as the datum needed to fire it, but the patient record mentions only biliary surgery, it does not take a deep knowledge of medicine to fire that diagnostic rule. In both these cases domain knowledge is needed, but the reasoning involved is not diagnostic reasoning in our specific technical sense. One can imagine an expert diagnostician turning, in the course of her diagnostic reasoning, to a nurse in charge of the patient record and asking if there was evidence of anesthetic exposure or of abdominal surgery, and the nurse answering affirmatively in both the instances without his being trained in diagnosis at all.

When we faced this problem in the design of MDX, we realized that it would be very inelegant to combine reasoning of this type with the diagnostic reasoning that we had isolated as a specific type of problem-solving activity. We were led to the creation of a separate subsystem for managing patient data, much like the nurse alluded to earlier. For all questions concerning manifestations, MDX simply turned to this subsystem, which performed the relevant reasoning and returned the answer. We were surprised to discover that all the retrieval activities of this "data base assistant" could be captured in a uniform paradigm, to be elaborated

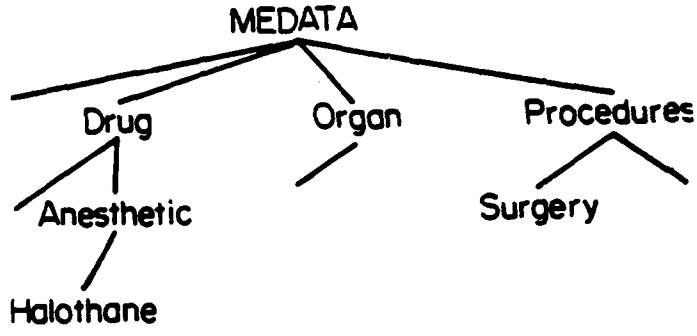


Figure 2.

shortly. Mittal (Mittal, 1980) describes this in detail as do the references (Mittal, Chandrasekaran, 1981) and (Mittal, Chandrasekaran, 1969). Similar to our discussion regarding the diagnostic task, we just touch upon the issues here sufficient to make our main points regarding decomposition.

This data base–called PATREC–is organized as a hierarchy of medical data concepts. A fragment of the hierarchy is shown in Fig. 2.

At a representational level, there is nothing novel here: each medata concept is represented as a frame, and the inference rules that we will describe shortly are implemented as "demons" or "procedural attachments." However what will be worth noticing is the fact that all these rules will be of a certain uniform type. For the purpose of illustration, let us consider the SURGERY concept. SURGERY frame has LOCATION and PERFORMED? slots, among others. The "PERFORMED?" slot has the following rules:

1. If no surgery in the enclosing organ, surgery not done.
2. If surgery in a component, infer surgery in this organ.
3. If no surgery in any of the components, then infer no surgery in this organ.
4. If evidence of anesthetic, infer "possibly."

The DRUG frame has the following rules in the GIVEN? slot:

1. If any drug of this type given, infer this drug also.
2. If the drug class was not given, rule out this particular drug.
3. If all *drugs* of this type were ruled out, rule out the class too.

These rules need not be attached to the successors of DRUG, since they can inherit these rules–this is a fairly standard thing to do in frame-based systems. A successor may have further rules which are particular to it, e.g. the ANESTHETIC concept has the rule:

If major surgery, infer ANESTHETIC given, possibly.

Let us reemphasize that the interesting thing about the system is not

rare knowledge base system that doesn't—but that it is a collection of conceptual specialists tuned to a particular type of problem-solving. All the embedded inference rules have a

common structure: derive the needed data value from data values relating to other concepts. The inferential knowledge that is encoded in the concepts is specific to the data retrieval task in a data base activity.

Let us consider some examples. Suppose the stored datum is that "Patient was given halothane." The HALOTHANE frame now has its GIVEN? slot filled with "Yes." Consider the following series of questions:

Q1. Given Anesthetic
A : YES

(ANESTHETIC specialist inherits the rules from the DRUG frame. Rule 1 generates the question, among others. "Given Halothane?" "Yes" is propagated upwards.)

Q2. Any Surgery performed?
A : Possibly

(SURGERY specialist fails with rules 1, 2 and 3. Rule 4 places query "Given Anesthetic?" to ANESTHETIC specialist. "Yes" answer results in "Possibly" to Q2. This is an example of lateral inheritance.)

Similarly if the stored datum were "Patient had major surgery," and the query were, "Given Anesthetic?", rule 1 in ANESTHETIC would have given the answer "possibly."

Another more complex example of data retrieval reasoning by PATREC is the following:

DATA: A liver-scan showed a filling defect in the left hepatic lobe. The liver was normal on physical exam.
Q : Liver Normal?
A : No

(On liver-scan data, the following chain of inference took place: (a) filling-defect in lobe → lobe not normal; (b) If <comp-of> liver not normal → liver not normal. On the other hand, Physical examination produced "Normal" as answer. By using a general principle that when there are contending answers, *non-default* value should be chosen—the default for "Normal?" slot of LIVER is "Yes"—the answer "No" was generated.)

The main points relevant here are, as in the case of the diagnostic task: (1) There is no separation between a knowledge base and a problem-solver. Problem-solving is embedded in the knowledge structure. (2) All the conceptual specialists perform the *same type* of problem-solving, in this case, inheritance of data from other specialists. (3) Concepts with the same name, say LIVER, in the diagnostic structure and the data retrieval structure have different pieces of knowledge and do different things. This is akin to the fact that the LIVER concept of a diagnostician is bound to be different from that of the data base nurse. The concepts in this sense are "tuned" for different types of knowledge use.

What-Will-Happen-If (WWHI) Or Consequence Finding

We said that among the many types of problem-solving that take place in a knowledge-rich domain is that of answering questions of the form "What will happen if X is done?" Examples are: "What will happen if valve A is closed in this power plant when the boiler is under high pressure?"; "What will happen if drug A is administered when both hepatitis and arthritis are known to be present?" Questions such as this can be surprisingly complex to answer since formally it involves tracing a path in a potentially large state space. Of course what makes possible in practice to trace this path is domain knowledge which constrains the possibilities in an efficient way.

The problem-solving involved, and correspondingly the use of knowledge in this process, are different from that of diagnosis. For one thing, many of the pieces of knowledge for the two tasks are completely different. For example, consider answering the question in the automobile mechanic's domain: "What will happen if the engine gets hot?" Looking at all the diagnostic rules of the form, "hot engine → <malfunction>" will not be adequate, since <malfunction> in the above rules is the *cause* of the hot engine while the consequence finding process looks for the effects of the hot engine. Formally, if we regard the underlying knowledge as a network connected by cause-effect links, where from each node multiple cause links as well as effect links emanate, we see that the search processes are different in the two instances of diagnosis and consequence-finding. The diagnostic concepts that typically help to provide *focus* and constrain search in the pursuit of correct causes will thus be different from the WWHI concepts needed for the pursuit of correct effects.

The embedded problem-solving is also correspondingly different. We propose that the appropriate language in which to express the consequence-finding rules is in terms of *state-changes*. To elaborate:

1. WWHI-condition is first understood as a state change in a subsystem.
2. Rules are available which have the form "<state change in subsystem> will result in <state change in subsystem>". Just as in the case of the diagnosis problem, there are thousands of rules in the case of any nontrivial domain. Again, following the diagnostic paradigm we have already set, we propose that these rules be associated with *conceptual specialists*. Thus typically all the state change rules whose left hand side deals with a subsystem will be aggregated in the specialist for that subsystem, and the right hand side of those rules will refer to the state changes of the *immediately affected* systems.

Again we propose that typically the specialists be organized hierarchically, so that a subsystem specialist, given a state change to it, determines by knowledge-based reasoning the state changes of the immediately larger system of which it is a part and calls that specialist with the information determined by it. This process will be repeated until the state change(s) for the overall system, i.e., at the most general relevant level of abstraction, are determined. This

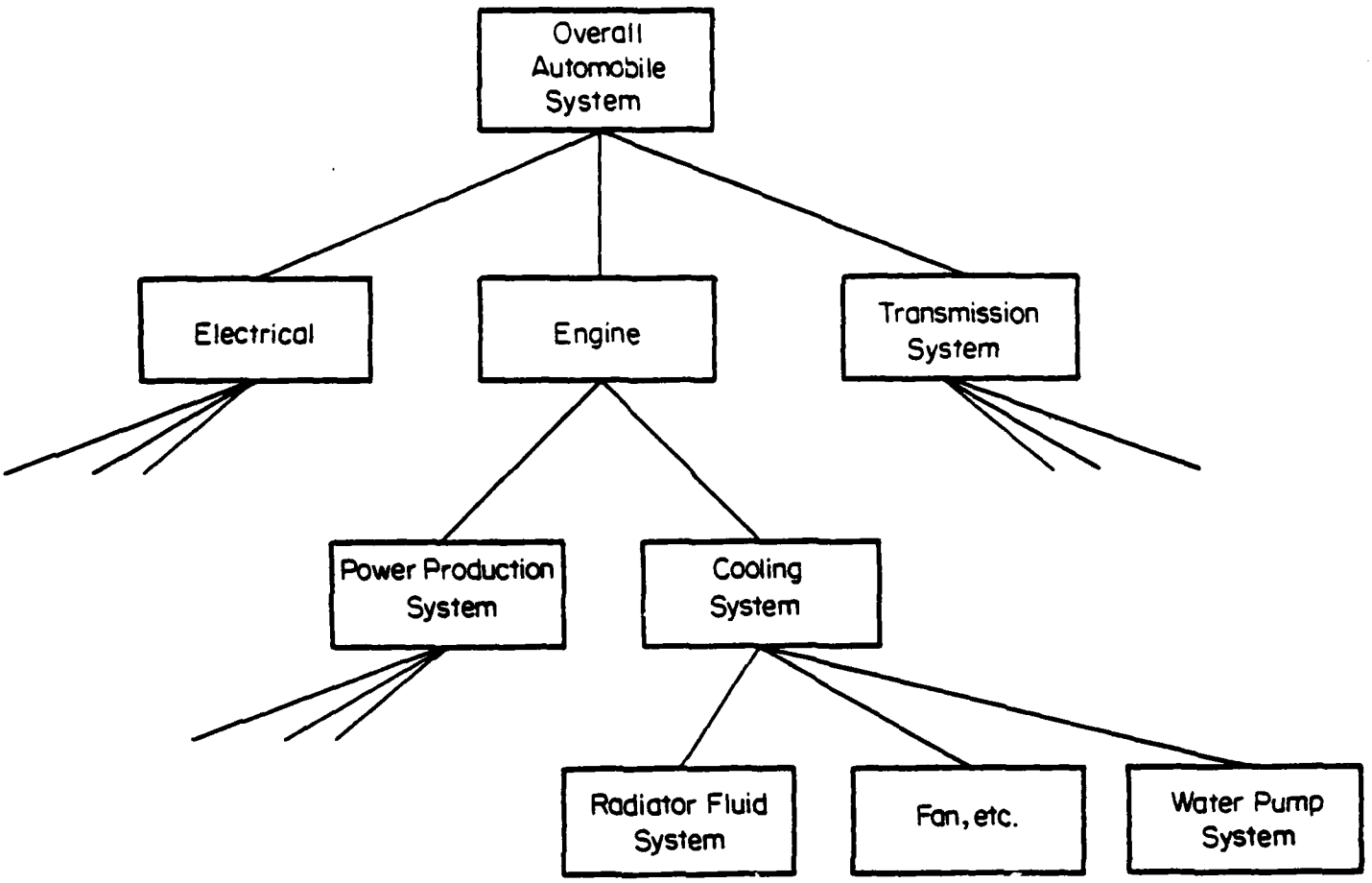


Figure 3.

form of organization of the rules should provide a great deal of focus to the reasoning process.

An Illustrative Example. Consider the question, in the domain of automobile mechanics, "WWHI there is a leak in the radiator when the engine is running?" We suggest the specialists are to be organized as in Fig. 3.

The internal states that the *radiator fluid subsystem* might recognize may be partially listed as follows: {leaks/no leaks, rust build-up, total amount of water,...}; similarly, the *fan subsystem* specialist might recognize states {bent/straight fan blades, loose/tight/disconnected fan belt,...}. The *cooling system subsystem* itself need not recognize states to this degree of detail: being a specialist at a somewhat higher level of abstraction it will recognize states such as {fluid flow rate, cooling-air flow rate...etc.}. Let us say that the *radiator fluid specialist* has, among others, the following rules. The rules are typically of the form:

<internal state change> → <supersystem state change>

leak in the radiator → reduced fluid flow-rate
high rust in the pipes → reduced fluid flow-rate
no antifreeze in the water
and very *cold weather* → *zero fluid flow* etc.

The cooling system specialist might have rules of the form:

low fluid-flow rate and engine running → engine state hot
low air-flow rate and engine running → engine state hot

Again note that the internal state recognition is at the appropriate level of abstraction, and the conclusions refer to state changes of its parent system.

It should be fairly clear how such a system might be able to respond to the query about radiator leak. Again a blackboard for this task would make it possible to take into account subsystem interaction.

Unlike the structures for the diagnostic and data retrieval tasks, we have not yet implemented a system performing the WWHI-task. While we cannot speak with assurance about the adequacy of the proposed solution, we feel that it is of a piece with the other systems in pointing to the same set of morals: embedding still another type of problem-solving in a knowledge structure, which consists of cooperating specialists of the same problem-solving type.

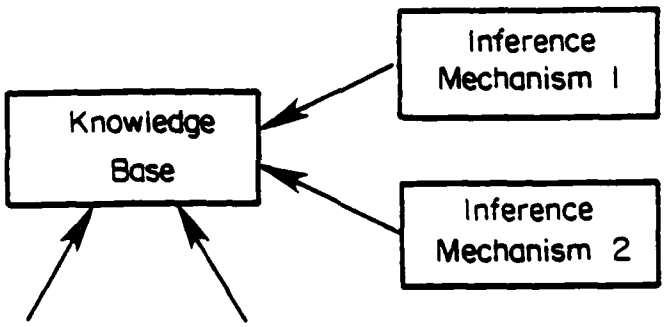


Figure 4.

Knowledge-Use Taxonomy

There has been a growing realization in the field that the important issue in knowledge systems is to determine how knowledge is to be used. Our foregoing examination of the three tasks—each of which is not some ad hoc need for medical reasoning, but is a generic task that arises in a number of domains—leads us to propose the following theses.

1. There is taxonomy of problem-solving regimes that are involved in expert problem-solving. We have identified three members of this taxonomy
   - diagnostic (classificatory): establish-refine, top-down.
   - consequence-finding: abstract state from low-level description to higher-level description, bottom-up.
   - data retrieval: inheritance/inference of values from data values in other concepts.

   There are obviously more. Our research is oriented towards finding more elements of this taxonomy and determining their interrelationships.
2. For each type of problem-solving there is a separate knowledge structure, with the associated p.s. regime embedded in it. Thus a domain of knowledge can be decomposed into a collection of structures, each of which specializes in a p.s. type. We can call this a horizontal decomposition of the domain.
3. Each of the structures in (2) above can be further decomposed into a collection of specialists, all of whom are of the same p.s. type, but differ from each other in the conceptual content. We have indicated how this decomposition can be done for the three tasks considered. We term this decomposition a vertical decomposition.

A Paradigm Shift

The prevalent approach to knowledge base systems is based on the decomposition in Fig. 4:
In this paradigm, knowledge representation is separated from its use. This approach has the attraction of generality and a certain kind of modularity.

The representational questions are dealt with in this approach in a manner to satisfy the criterion of expressiveness, or so-called epistemological adequacy of McCarthy (McCarthy, Hayes, 1969). The efficiency responsibilities are put on the shoulders of the inference mechanisms: they have to have the so-called heuristic knowledge in order to use the knowledge efficiently for problem-solving. Our approach is based on a rather different decomposition of the same problem, as indicated in our discussion on horizontal decomposition in the previous section.

Pictorially, the viewpoint of knowledge-based systems that we advance can be given as Fig. 5.

Thus the overall knowledge system is viewed as a *collection of specialists in inference types*, who cooperatively solve a given problem. While in the figure we have indicated the communication among these specialists to be unconstrained, in fact, however, it may not be so. There may be reasons why only certain problem-solving specialists can talk to other problem-solving specialists. This is an open research problem in our approach.

Production Rule Methodology. In most of the preceding discussions the *representation* of knowledge has been in the form of rules. We feel that this is not accidental, but that rules represent a basic form of cognition, viz., "how-to" knowledge. This was recognized early in AI by Newell and Simon (Newell, Simon, 1972) who named the rules *production rules*. Later, the Stanford Heuristic Programming Project and others extended this production rule methodology for a wide class of expert system design problems. We are thus in agreement with the use of rules as a basic knowledge representation formalism in expert systems.

There are two aspects in which our methodology differs from current work on rule based systems. We have already alluded to the difference in the viewpoint which regards knowledge not as an independent structure to be used by different problem-solvers, but as *embodiments* of implicit problem solving knowledge. Related to that is the idea that the central determinant of effective use of knowledge is how it is *organized*. Our approach begins to provide criteria for performing the organization of a complex body of knowledge. It is well-known that production rules need to be organized not simply for purpose of efficiency, but for *focus* and *control* in problem-solving (see (Lenat, Harris, 1978) for a discussion of these issues). We are proposing two *organizing constructs*, which extend the production rule methodology to make it applicable to a larger class of problems. One construct is the problem-solving regime and the other is that of a *conceptual specialist*.

Related to these organizational notions is the other aspect of the difference between our approach and the current production rule methodologies. We do not use uniform problem-solving mechanisms (backward chaining, e.g.) across the whole domain. As indicated, the problem-solving method differs from knowledge structure to knowledge structure.

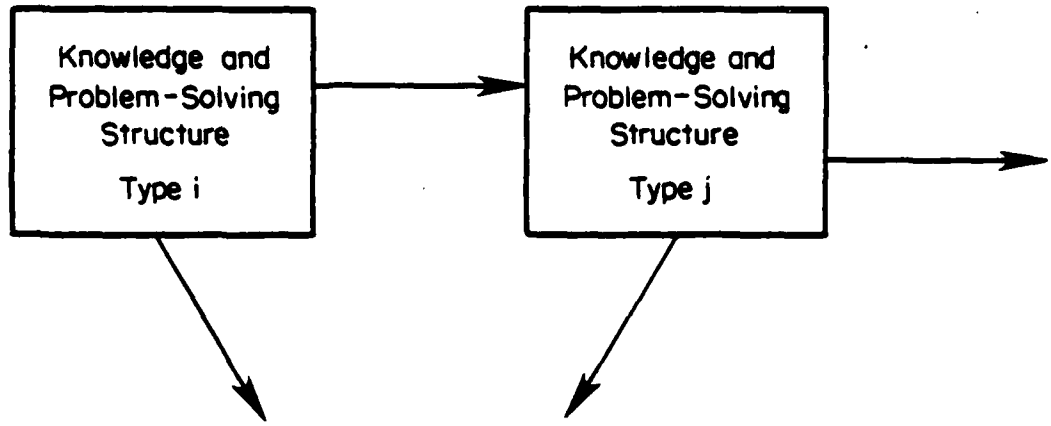


Figure 5.

Role of "Deep" Models

Deep and Compiled Structures. Recently Hart (Hart, 1982) and Michie (Michie, 1982) have written about the "depth" at which knowledge is represented and used in problem solving by expert systems. Distinctions such as "deep" vs "surface" and "high road" vs "low road" have been made in this connection. There is no clear definition of what constitutes a deep model - in fact precisely that issue is an open area of research in the field, but the intuition is that it models the underlying processes of the domain. Michie remarks that most expert systems that are extant don't have deep models in this sense, but instead can be viewed as a data base of patterns with a more or less simple control structure to navigate through the data base. It is argued that surface systems of this type have inherent limitations in hard problems, and that a system which has a deep model will be able to turn to it when faced with an especially knotty problem, much like a human expert might resort to "first principles" in a similar situation. In addition to deep models of the domain, the human problem solver also uses other sorts of knowledge such as "common sense" knowledge of various kinds.

In the rest of the discussion in this section we will explicitly consider the diagnostic task only. But the arguments will apply to other tasks as well.

We argue in (Chandrasekaran, Mittal, 1982) for a thesis which might at first sound counter-intuitive. Let us assume that we wish to design a diagnostic system in a particular domain. Let us further assume that we can successfully construct a deep model of the domain, and also specify the problem solving processes that will operate on that model. The thesis that we argue in (Chandrasekaran, Mittal, 1982) is as follows. Between the extremes of a data base of patterns on one hand and representations of fdeep knowledge (in whatever form) on the other, there exists a knowledge and problem solving structure - along the lines outlined in the section on the diagnostic task in this paper - which (1) has all the relevant deep knowledge "compiled" into it in such a way that it can handle all the diagnostic problems that the deep knowledge, if explicitly represented and used in problem-solving, can handle; and (2) will solve the diagnostic problems more efficiently than the deep structure can: but (3) it cannot solve other types of problems- i.e., problems which are not diagnostic in nature - that the deep knowledge structure potentially could handle. The argument is rather detailed, but the essence of it consists of analyzing the ways in which the diagnostic structure may fail to solve a particular problem, and tracing that failure to either missing knowledge in the deep model itself or in the problem solving processes that operate on it. Thus the range of *diagnostic* problems that can be solved with the deep model is exactly coextensive with the problems solvable with the diagnostic problem solving structure that can be derived from it.

There is another way of looking at this. There is a natural decomposition in the problem solving responsibilities between the underlying knowledge structures and the diagnostic structure. The former builds the diagnostic structure and the latter solves specific diagnostic problems. Human experts often resort to deep models because the diagnostic structures are in general incomplete. This decomposition also translates into a natural division of responsibility for explanation of decisions. See (Chandrasekaran, Mittal, 1982) for more discussion on this.

Multiple Uses of Knowledge. It is possible that there will be some redundancy in knowledge represented in our approach, since it calls for knowledge to be encoded in a problem solving structure according to its usage - some pieces of knowledge may appear in several structures. (See comments in (Gomez, Chandrasekaran, 1981) on redundancy and biasing of knowledge.) Is this a good thing?

We have a choice: (1) We can have the knowledge in a deep enough form, but as, say, a diagnostic problem presents itself, we can first generate fragments of diagnostic knowledge as needed and use it to solve the given problem. Similarly for a WWHI problem, etc. Or, (2) we can choose the tasks to be experts in , compile the problem solving structures for them, accepting some redundancy. The latter is faster for those tasks for which they are designed, the former is more economical in storage. A classic trade-off!

In a sense the former situation describes, e.g., a bright medical school graduate who has a functional understanding of the phenomena of the human body, but that knowledge is not yet molded into effective problem solving structures of particular types. We suspect that what happens even among experts is that they build powerful problem solving structures to account for a good portion of foreseeable situations, and thus need to resort to the deeper structures only for the harder problems. This is a compromise between the requirements of expertise and memory.

The Nature of the Deep Model

There is an additional problem with option 1 in the current state of the art: we don't know how to do it! This requires an adequate theory of the nature of the deep model. When a person newly understands how a device works, e.g., it is doubtful that what he has acquired is merely a collection of rules or facts, or a network of causal relations. One can have all these and still not "understand." The sense of understanding must correspond to some *organization* of these pieces of knowledge for some class of purposes. The organization must be such that it can be processed to produce problem solving structures for various tasks. The nature of the deep model is an extremely important area of research. The work of (Rieger, Grinberg, 1976), (Pople, 1982), (Patil, 1981) and (de Kleer, Brown, 1982), to name a few researchers who have looked at this problem, seem very relevant here. However, in order to adequately represent knowledge at this level, notions of an organizational nature particular to that level also seem important.

On Hierarchies

In all the tasks that we considered in this paper, the knowledge structures were strongly hierarchical. While hierarchical organizations have a strong intuitive appeal, in AI there is also a strong tradition of "heterarchies" and network structures. Difficulties with hierarchical classification structures have been noted in (Fahlman, et al, 1981). Also concerns such as "the world is not hierarchical" are voiced in response to proposals for hierarchical organizations.

This is not the place to discuss the important issue of hierarchical structures in problem solving. The following brief remarks should suffice for our purposes. First of all, the main thesis about decomposing knowledge by problem solving types and embedding of the problem solving in the knowledge sources is itself independent of whether the structures for a problem solving type are hierarchical. Secondly, our general strategy has been to start by looking for hierarchical decompositions, and where there seems to be a need for communication outside of the hierarchical channels, to provide it in a carefully controlled fashion such as the blackboards discussed in (Gomez, Chandrasekaran, 1981). (See (Chandrasekaran, 1981) for a discussion of different kinds of communication needs in a distributed problem solving situation.) For example, in (Gomez, Chandrasekaran, 1981) we discuss how certain kinds of relations between disease hypotheses belonging to different portions of the hierarchy – such as disease A being secondary to disease B – can be handled within a hierarchical framework by the use of blackboards. Finally, it ought to be stated clearly that hierarchies are not "out there," but imposed by the thought processes for control over problem solving. Thus it is a powerful weapon, but by no means a sufficient one. It will be rash to conclude that all complex problem solving in all complex domains can be crisply conducted in a single hierarchical framework. Reasoning about feedback and reasoning with multiple perspectives are two examples where additional machinery seems to be needed beyond the hierarchical framework.

The Organization of the Medical Community

Evidence of Horizontal Decomposition. The medical community collectively is a good case study in the principles by which knowledge may be structured for cooperative, effective problem-solving. Corresponding to our notion of horizontal decomposition along the lines of problem-solving types, we can identify clinicians, educators, pathologists, radiologists, medical records specialists, etc. Clinicians combine the diagnostic and predictive knowledge structures, for practical reasons having to do with the close interaction between diagnosis and therapy. Medical record specialists, as their name indicates, serve to organize patient data and retrieve them effectively. Radiologists are not diagnosticians in the same sense as clinicians are: their problem-solving is to reason from imaging descriptions to confirm or reject diagnostic possibilities; they are largely perceptual specialists.

Evidence of Vertical Decomposition. Corresponding to our vertical decomposition, many of the above problem-solvers are organized into conceptual hierarchies. For instance, an internist is the top-level diagnostic specialist, who may call upon liver or heart specialists for further investigation of a problem. The top-down problem-solving for diagnosis is indicated by the fact that a sick person typically first goes to an internist, who may refer the patient on to more detailed specialists.

Evidence for Embedding Problem-Solving. If the medical community were organized according to the currently accepted paradigm in expert systems, i.e., a common knowledge base shared by different problem-solvers who themselves are without domain-knowledge, one would expect to have knowledge-specialists, who would be rather like encylopaedias, and problem-solving specialists who would possess expert-algorithms for, say, diagnosis, without any medical knowledge about particular medical concepts. Thus whenever a patient came, the diagnostic specialist would consult the knowledge-base specialist and together they would arrive at a diagnostic conclusion.

However, that is not the way the community works. Instead we find that experienced medical specialists possess expertise which is not a raw knowledge-base, but which is highly effective in problem-solving. On the other hand, a medical student without clinical experience is more like a pure knowledge-base. As he or she becomes more experienced in various types of problem-solving, the unstructured knowledge base slowly begins to shape and structure itself, so that pieces of knowledge are tuned for ready and effective use.

Acknowledgments

I owe a great debt of intellectual gratitude to Fernando Gomez, who was responsible for many of the central theoretical ideas behind MDX. Sanjay Mittal has been a dedicated co-worker, responsible for much of the implementation as well as for many of the ideas incorporated in PATREC and RADEX. Jack Smith, M.D., is, in addition to being the medical component of our group, also a skilled computer scientist, which contributed greatly to a smooth AI/medical interface. I thank Lee Erman who read a draft of this paper and made several suggestions to improve its clarity. Needless to say his help does not necessarily imply endorsement of all the ideas in this paper.

The preparation of this paper was supported by NSF grant MCS-8103480 and AFOSR grant 82-0255. The computing resources of Rutgers University Laboratory for Computer Science Research were used extensively in system development, and this was made possible by the Biotechnology Resources Division of NIH, under grant RR-00643.

References

Chandrasekaran, B. (1981) Natural and social system metaphors for distributed problem solving: Introduction to the issue. *IEEE Transactions on Systems, Man & Cybernetics* SMC-11(1):1-5.

Chandrasekaran, B. and Mittal, S. (1982) Deep versus compiled knowledge approaches to diagnostic problem-solving. *Proc. of AAAI*, 349-354.

Chandrasekaran, B., Mittal, S., and Smith, J. (1980) RADEX—Towards a computer-based radiology consultant. In E. S. Gelsema and L. N. Kanal (Eds.), *Pattern Recognition in Practice*. Amsterdam, North Holland.

Chandrasekaran, B., Mittal, S., and Smith, J. W. (1982) Reasoning with uncertain knowledge: the MDX approach. *Proceedings of the 1st Annual Joint Conferences of the American Medical Informatics Association.*

Erman, L. D. and Lesser, V. R. (1975) A multi-level organization for problem-solving using many diverse cooperating sources of knowledge. *IJCAI 4*, 483-490.

Fahlman, S. E., Touretzky, D. S., and van Roggen, W. (1981) Cancellation in a parallel semantic network. *IJCAI 7.*

Gomez F. and Chandrasekaran, B. (1981) Knowledge organization and distribution for medical diagnosis. *IEEE Transactions on SMC*, SMC-11:34-42.

Hart, P. E. (1982) Direction for AI in the eighties. *SIGART Newsletter*, 79:11-16.

de Kleer, J. and Brown, J. S. (1982) Foundations of envisioning. *Proc. of AAAI*, 434-437.

Lenat, D. B. and Harris, G. (1978) Designing a rule system that searches for scientific discoveries. In D. A. Waterman and F. Hayes-Roth, (Eds.), *Pattern-Directed Inference Systems*. New York: Academic Press.

McCarthy, J. and Hayes, P. J. (1969) Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, Vol. 4, New York: Elsevier.

Michie, D. (1982) High-road and Low-road programs. *AI Magazine*, 3:21-22.

Mittal, S. (1980) *Design of a distributed medical diagnosis and database system.* Doctoral Dissertation, Dept. of Computer and Information Science, The Ohio State University.

Mittal S. and Chandrasekaran, B. (1981) Conceptual representation of patient data bases. J. Medical Systems.

Mittal, S. and Chandrasekaran, B. *(1981) Software design of* knowledge-directed database systems. *Proc. 1st Conf. Foundations of Software Technology and Theoretical Computer Science,* Tata Institute of Fundamental Research, Bombay, India.

Mittal, S. and Chandrasekaran, B. (1981) Some issues in the design of knowledge-directed databases. Working Paper, AI Group, Dept. of Computer and Information Science, The Ohio State University.

Newell, A. and Simon, H. A. (1972) *Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall.

Patil, R. S. (1981) *Causal representation of patient illness for electrolyte and acid-base diagnosis.* Doctoral Dissertation, TR-267, MIT Lab for Computer Science, Cambridge, Mass.

Pople, H. E. (Forthcoming) Heuristic methods for imposing structure on ill-structured problems. In P. Szolovits (Ed.), *Artificial Intelligence in Medicine*. Westview Press.

Rieger, C. and Grinberg, M. (1976) The causal representation and simulation of physical mechanisms. Tech. Rep. TR-495, Dept. of Computer Science, The University of Maryland.

IEEE Computer Society, Trends & Applications Conf., May 1983, Gaithersburg, MD.

AN APPROACH TO EXPERT SYSTEMS FOR MECHANICAL DESIGN

David C. Brown*
B. Chandrasekaran
Artificial Intelligence Group
Dept. of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Abstract

We present an approach to expert systems for mechanical design called Design Refinement, which addresses a subset of design activity by using a hierarchy of conceptual specialists that solve the design problem in a distributed manner, top-down, choosing from sets of design plans and refining the design at each level of the hierarchy.

1. Introduction

In terms of economic impact, one of the most important areas for AI technology is CAD/CAM. AI is applicable to a variety of subtasks in CAD/CAM: process control and robotics are areas where work has already been done. However, in terms of intellectual difficulty the central problem is design itself. While much AI-related work is being done in the creation of design aids in the VLSI area [9,10,11], there has been relatively little attention paid to the knowledge structuring and problem solving issues in the main problem of design itself. This paper addresses the problem of expert systems for mechanical design. For an important class of design tasks, we present an approach with design refinement as the central problem solving activity. This activity can be quite complex, but our aim here is to provide a first-cut analysis of this process in order to show it's potential for generating design expert systems.

---

*Currently on leave from the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609

The creation of computer-based expert consultants in any area of human endeavour requires an analysis both of the knowledge structures that the corresponding human specialist uses, and the problem-solving methods that underlie the different tasks. Thus much of our discussion will be taken up with an account of these aspects of the design process.

2. Types of problem solving

We have been developing a framework for decomposing a complex body of knowledge into small knowledge sources, and organizing them into a structure of specialists engaged in collective problem solving. We have identified several distinct types of problem solving that are useful in the design of expert systems [3]. One advantage of this is that it enables one to characterize which kinds of expert problem solving we know how to mechanize.

One type of problem solving is capable of performing diagnosis, i.e., capable of reasoning about how to classify a complex description of reality as a node in a diagnostic hierarchy [1]. Another type of problem solving (called WWHI by us) is capable of reasoning about consequences of contemplated actions on complex systems, such as answering the question, "What will happen if I close that valve in this power plant?". We believe that design can be classified as a distinct type of problem solving, and that it is different from the other types we have identified. We will outline how a community of design specialists can work together to convert a list of specifications for a component into a detailed design for that component.

3. Discussion of design activity in general

In general, design is a highly creative activity, the underpinnings of which we in AI only dimly perceive. Often the design goals themselves are only vaguely specified, and the feedback from attempts to achieve these goals serves to modify them. Thus designers of new systems work with knowledge structures and problem solving techniques that we cannot yet adequately capture with AI technology. What the current state of the art in

AI can do for them is more along the lines of support systems (intelligent graphic aids, knowledgeable data bases, etc.), rather than actually performing the design task itself [13,5].

In a typical industrial operation, however, the daily task of the designer is frequently less exalted than the kind of highly creative design mentioned above. In fact, most industries perform design tasks which, for the purposes of the current discussion, can be classified into roughly three categories (to simplify what really is a spectrum from the most open ended to the most routine).

Class 1 design is done so rarely it is often the basis of a new company, division or a major marketing effort. Major inventions belong to this category. The design activity in this case involves access to wide-ranging knowledge structures, and searches in a very large space of design alternatives.

Class 2 design is closer to the routine, but still many of the established patterns may be broken. Some aspects of design may require substantial innovation; e.g., in a company which manufactures integrated sensor systems for control of sheet processes, the need to take into account extremely hot ambient conditions for a particular order may require suspension of the standard design and launching of an investigation about new (for the company) techniques of control of ambient temperature within the sensor housing, which might in turn result in new sealing techniques and so on. In many companies, this happens periodically, but is undertaken with the hope that the investment of time and energy will be paid off by identifying a potentially large market, in which the new elements of design can be "routinized".

Class 3 design, the most routine, follows a set of relatively well-established design alternatives which are reasonably well-understood, but nevertheless still require a well-trained human expert to perform the design task. We do not intend to imply that the task is "simple"; in fact, we cannot fully substitute for the human expert, and new advances in AI are called for. For example, simple approaches, such as use of a data-base of design parameters with associated designs, may work for some problems, but in general they will fail due to the large number of combinatorial possibilities.

Let us examine in some further detail the nature of Class 3 design tasks. There exists a large number of industries which make one-of-a-kind products, where each is of the same general class. For example, AccuRay Corporation, our collaborator, designs and delivers control systems to industries which manufacture sheet products (aluminum foil, paper, etc.). These control systems have sensors which continually monitor various properties of the sheets, and provide the signals which can be used to keep the thickness within certain bounds. However, even within one industry, no pre-constructed system can be placed within a functioning mill. Each control system is built anew from specifications that are gathered from a particular prospective installation. The control system itself is complex and consists of the sensor assembly (frame, carriage, sensors, etc.) and the complex computing system (minicomputers and software) that goes with the sensor hardware.

The complexity of the task arises from the numerous subcomponents and their sub-subcomponents, each of which needs to be specified according to top-level specifications. The top-level specifications of two different plants in the same industry differ considerably, and this adds to the complexity. For example, no two paper mills are alike, they themselves having been designed to the differing specifications that arise from the differing products and markets of the companies. Numerous constraints exist among the parameters of the subcomponents, contributing further to the complexity of the task.

While a Class 3 design may still be conceptually complex, the design alternatives at each stage are not as open-ended as in some of the stages of Class 2 or Class 1 designs, nor is there the vagueness and nonoperationalism of the top-level goals or the difficulty with identification of constituent substructures that is characteristic of Class 1 design. That is, despite complexity, the design is intellectually more manageable, and the variety of knowledge sources that are accessed during the execution of the task are relatively small and can be identified in advance.

Sometimes, however, a design that had been classified as Class 3 might turn out during the design process to possess many Class 2 features. This happens if the design alternatives for a certain stage all fail, and an exploratory search

for a completely new alternative needs to be launched. Identifying a design task in advance as Class 3 is a nontrivial problem. But generally, experienced designers can judge if a project is Class 3 or not.

We propose that there is a very specific type of problem solving associated with expert design activity, especially of the Class 3 type: **a hierarchy of conceptual "specialists" solve the design problem in a distributed manner, top-down, by choosing from a set of design plans and refining the design at each stage.** Each refinement is done by a specialist calling its subspecialists in the hierarchy.

4. Class 3 Design

4.1 Description of Class 3 Design

In general, the component to be designed will be quite complex, with its own subsystems and substructures. We propose that the expert system to design the component be conceived as a hierarchical collection of *design specialists*, where the top levels of the hierarchy are specialists in more global subsystems of the component, while those at the lower levels deal with more specific subsystems or parts. They all access a design specification data-base. Intelligent data-base assistants can play an important role here; for a discussion see [2].

At each stage a specialist S has several prioritized alternative design plans. The specialist begins by inheriting some design parameters from its parent specialist, and it obtains relevant specifications from the data-base. Each of the plans can take these data as arguments. Parts of a plan may indicate immediately that constraints cannot be satisfied. This is considered as 'failure'. Parts of a plan access functions which can fill in the design templates independently, parts produce further values or constraints to be passed on to particular successors, while other parts of a plan give specific sequences in which the successors may be invoked. Thus, S fills in some of the design, then calls its successors in a given order with requests for refinement of the design of a substructure. If

some of the substructures are independent of each other, then subspecialists may be invoked in parallel. The overall global plan of the specialist prioritizes each subplan, invokes alternate plans in case of failure by one of the subspecialists, etc. When a specialist receives failure from one or more of its successors for all its plans, or when failure for given constraints can be deduced immediately, the specialist communicates that to its parent. The exceptions to this design refinement structure are the tip node specialists who can only fill in the design details. Typically as one goes down in the hierarchy, there are fewer alternative plans, and the plans themselves becomes more straightforward.

Let us consider a concrete, but highly simplified example for illustrative purposes -- the design of a Small Table consisting of a circular Top and a cylindrical Support. In a design specification the user may specify to the design system the materials to be used, or the required dimensions. The hierarchy of specialists for the expert system to design the Small Table is shown in figure 1. Note that the hierarchy need not be a strictly component-subcomponent hierarchy, and could be organized according to function.

System Organization:

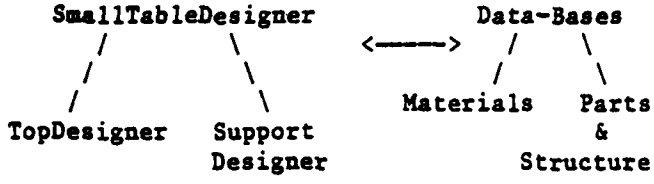


Figure 1 : Hierarchy of Specialists

The design process starts at SmallTableDesigner, at the point where the overall requirements are given to the design system. Consider the case in which SmallTableDesigner chooses its first plan, calls TopDesigner, which in turn also chooses its first plan, does the design of the Top, and returns the dimensions to its parent. Now

SmallTableDesigner calls SupportDesigner. Now suppose that this specialist's only plan fails to generate a successful design within the constraints; i.e., the strength requirement and dimension constraints are not reconcilable according to its expertise. This would cause 'failure' to be returned to SmallTableDesigner. SmallTableDesigner then calls TopDesigner again with a further constraint about the weight that is permitted. Now TopDesigner will invoke its second plan (which is a more expensive plan to execute), and some information about the new Top design is passed to SupportDesigner through its parent SmallTableDesigner, causing SupportDesigner to succeed, and the design to succeed.

An important thing to note is that very large numbers of designs are encoded in an economical way in this approach. While plans are "pre-compiled", actual designs aren't -- they are actually generated during problem solving. Further, the expertise of each specialist can be selectively increased by carefully integrating new plans into the specialist. Finally, the human designer can find causes of failure in the feedback from the expert system, and, for example, might be able to come up with a "new" way to design the support, so that the rest of the system can proceed on a more automatic design.

What makes Class 3 but not Class 2 or Class 1 amenable to this approach is the fact that in Class 3 projects, a clear idea of the identities of the specialists in the hierarchy is available (in Class 1, one doesn't even know who the successors might be for a specialist), and further, the alternative plans of each specialist can be identified and are relatively small in number (in Class 2, the needed alternative design plans are not specified).

4.2 The role of analytic routines

The image of the designer sitting in front of the CAD graphics terminal, running stress analysis routines and visually inspecting the stress patterns of a component is typical in descriptions of how CAD systems work. Analysis of design is an intrinsic part of the total design process, but what role does such analysis play in expert systems for Class 3 design?

The preceding description of how the plans work has been at too high a level to bring out this aspect. In fact, analysis of design plays a role in several steps of the process. When a specialist inherits design constraints from its parent, accesses specification data from the data base and decides if there are any obvious difficulties in proceeding with the design, one of the options will be to run some analysis packages. Similarly, at any stage in the choice of values for a design, the plan may call for some analysis. The only difference from the current CAD practice is that the analysis and evaluation will be under the control of the specialist's plans.

5. Prototype system

5.1 Introduction to Prototype System

A prototype design expert system has been implemented for the domain example used above -- that of a Small Table which consists of a circular Top and a cylindrical Support. As above, the design hierarchy consists of a *Small Table specialist that* uses a Top specialist and a Support specialist. The system has a small design data-base with information about dimensions of parts, the structure of the table, and the types of materials available for use. Figure 2 shows the information that a specialist has available.

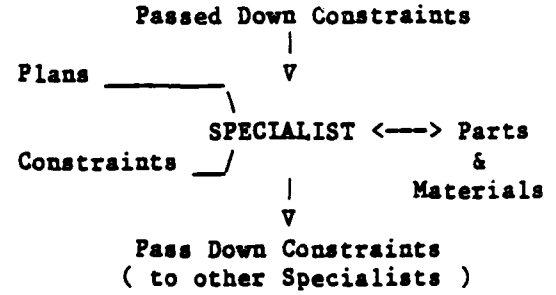


Figure 2 : Overview of Specialist

Each design specialist has a set of built-in constraints that specify what has to be true in order for that specialist to complete its design, and a collection of plans that can be selected for design, redesign, or verification of an existing design. Redesign is the alteration of an existing design due to the establishment of new specifications. As this is a class 3 task, each specialist has fixed plans that approach the design task at that level of the hierarchy in some prespecified manner. Each specialist has a 'plan suggester' that selects the plan to be executed.

The system is started by giving the Small Table designer the user's design specifications -- for example, that the top be Marble. When a specialist calls a sub-specialist all relevant constraints from the user's specification are passed down. A specialist succeeds when its selected plan succeeds, and that can only happen when its built-in and passed-down constraints are satisfied. The system uses default values to do 'rough' design, and can make small refinements to those values if necessary in order to satisfy constraints. The 'trace' of the system is very readable, with checking and decisions being handled in appropriate places. The flow of control in the system is well disciplined and it is clear at each step what part of the design is being attempted and why.

5.2 Structure

Specialists The specialists in the system are represented by a list of attribute-value pairs, plus some associated programs and constraints.

```
Specialist:
Name                 -- SmallTableDesigner
SpecialistsUsed      -- (TopDesigner
                            SupportDesigner)
BuiltInConstraints   -- (STHeight STTopDiam
                        STSupportTopDiamRatio)
DesignPlans          -- (STPlan1)
ReDesignPlans        -- (STRPlan1)
VerifyPlans          -- (STVPlan1)
PlanSuggester        -- STSuggester
```

Plans The plan-suggester's job is to select the appropriate plans to be followed during this invocation of it's specialist. In general, the plan selected could depend on the requirements given to the specialist, the request made (eg. design), the history of the design being attempted, and the current state of the design. Each specialist will have at least one plan. Below we present a typical simple plan for the SmallTableDesigner. Note that the steps marked "**" indicate places where the system will fail in an unrealistic manner, but a better plan would be too complex to present here.

```
ST Plan:
 - use TopDesigner.
 - did it succeed?
     - No, then plan FAILS.
 - use SupportDesigner.
 - did it succeed?
     - No, then plan FAILS. **
 - check to see if design meets
        table design constraints,
        and the user's constraints.
 - problems?
     -.Yes, then plan FAILS. **
     - if everything OK,
        then table is designed,
        and plan SUCCEEDS.
```

Constraints Three constraints from the TopDesigner are given below. They restrict the thickness of the top, the material of the top, and the weight of the top. Constraints provide a readable specification, explicit localized tests of consistency, and can be used to direct the flow of control in the hierarchy.

```
Constraints:
  TMaterialThickness
  ((Thickness Top) > (MinThickness
                          (MadeOf Top)))

  TMaterial
  ((MadeOf Top) OneOf (Values
                          (Wood Marble)))

  TWeight
```

```
((Area Top) * (Thickness Top)
 * (UnitWeight (MadeOf Top))
< 10)
```

Data-bases Each specialist has access to knowledge about parts and materials. For each value to be specified in the design some default knowledge is available. In the implementation they are functions giving either context-free or context-sensitive values.

```
Part:
  Name              -- Top
  MadeOf            -- Unknown
  Thickness         -- Unknown
  Diameter          -- Unknown
  DefaultThickness  -- DTThickness
  DefaultDiameter   -- DTDiameter
  DefaultMadeOf     -- DTMadeOf

Material:
  Name          -- Wood
  MinThickness  -- 0.40000000E-1
  UnitWeight    -- 4
```

5.3 Action of system

In this section we will present portions of the output produced by the prototype system, in order to show its action. Note that the user's responses appear after the ">> *" prompt, and that all other text is from the design expert system. The two initial constraints specify that the table top is to be less than 2 feet in diameter, and that it must be made of Marble. Having been presented with the design specification the system can proceed.

```
DESIGN SYSTEM PROTOTYPE   (March 83)
Name of most general
  design specialist is?
>> *SmallTableDesigner

Any initial constraints?
  Answer y or n or filename
>> *CONSTRAINTS.INIT
```

```
Reading constraints from file

Constraints:
  TopSize     ((Diameter Top) < 2)
  MarbleTop   ((MadeOf Top) <-- Marble)

Using specialist -- SmallTableDesigner
In Mode ---------- Design
From specialist --- TOP
In plan ---------- TOP
Passed down ------- (MarbleTop TopSize)

Testing passed-down constraints
  MarbleTop is setting Marble
         as value of MadeOf in Top
Passed-down constraints OK

Testing built-in Constraints
Built-in constraints OK

Suggesting Plan STPlan1
Executing plan STPlan1

  Start by designing the Top
```

Before selecting its design plan the SmallTableDesigner checked the constraints, and set the top's material to be marble. The plan starts by using the TopDesigner to design the top. Once in control, the TopDesigner checks the constraints, just in case immediate failure can be reported, and then proceeds to select a plan.

```
Using specialist -- TopDesigner
In Mode ---------- Design
From specialist --- SmallTableDesigner
In plan ---------- STPlan1
Passed down ------- (TopSize MarbleTop)

Testing passed-down constraints
Passed-down constraints OK

Testing built-in Constraints
Built-in constraints OK

Suggesting Plan TPlan1
Executing plan TPlan1

  Looking for unspecified values in Top
  Try reasonable values first
```

At this point the plan for the TopDesigner is being followed, and, as this is a "tip node" in the design hierarchy, it must attempt to supply the appropriate missing details of the design. The plan continues, below, by using "default" (ie. apparently reasonable) values for those dimensions of the top that are not yet specified. After that, the design of the top appears to be complete, so it is checked using the built-in and passed-down constraints. Notice that both of the user's initial constraints are relevant for the TopDesigner and, consequently, have been passed to it. Unfortunately one of the defaults chosen conflicts with one of the design requirements. That default value gets reduced by 1 inch, allowing all constraints to be satisfied , and the design to continue. It could be argued that the default values should have been chosen with the constraints 'in mind', but we feel that the method used fits better into our refinement theory, as the defaults provide a "rough" design which is subsequently refined by consideration of the constraints.

```
    Selecting 2 ft. as Top diameter
    Selecting 0.2 as Top thickness

  Now TPlan1 checks for conflicts
  Testing passed-down constraints
    Constraints failing
      TopSize:  ((Diameter Top) < 2)
      Setting 1.9166669 as
           value of Diameter in Top
  Passed-down constraints OK

  Testing built-in constraints
  Built-in constraints OK

No conflicts found by TPlan1
Leaving plan TPlan1
Reporting Success of TPlan1 and TopDesigner

State of design:
Name             -- Top
MadeOf           -- Marble
Thickness        -- 0.19999999
Diameter         -- 1.9166669
DefaultThickness -- DTThickness
DefaultDiameter  -- DTDiameter
DefaultMadeOf    -- DTMadeOf
```

```
Name              -- Support
MadeOf            -- Unknown
Length            -- Unknown
Diameter          -- Unknown
DefaultMadeOf     -- DSMadeOf
DefaultLength     -- DSLength
DefaultDiameter   -- DSDiameter
```

At this point the SmallTableDesigner's plan calls for the design of the support, and will pass control to the SupportDesigner specialist, which proceeds in much the same way as above. Notice that here the default is context-sensitive. Note too that the SupportDesigner uses the services of a module not in the design hierarchy in order to test the strength of the support. After the support has been designed, the SmallTableDesigner checks the contraints again, and, as there are no problems, the plan is completed and the design is successful.

```
Next design the support
      .
      .
Selecting Metal as Support material,
   as Top material is Marble
      .
      .
Testing strength
      .
      .
Reporting Success of STPlanl
       and SmallTableDesigner
Result of Design attempt
  (SUCCEEDS)
```

5.4 Redesign mode

Suppose that, despite the TopDesigner having checked the weight of the Top to make sure that it wasn't too heavy, the SupportDesigner is unable to design a support that is strong enough. The SmallTableDesigner will ask the TopDesigner to redesign the top given this new information. In cases such as this we suspect that the specialist involved will be able to make a judgement as to whether this is really a request for a new design,

or a minor change to the existing design. Here, the TopDesigner would make a decision whether to select a design plan other than the one which has already been tried, or to select a redesign plan. A redesign plan will keep as much of the old design as possible and will concentrate on changing only whatever is necessary to correct the problem that the other specialist is having. Each specialist must keep or have access to a record of which plans have already been tried, under what conditions, and how successful they were.

6. AccuRay Research

The design refinement ideas presented in the previous sections are being used in an ongoing project to build an expert system for a more complex and realistic class 3 design task in an industrial environment. In conjunction with AccuRay we have studied the design of a small Air-cylinder (Figure 3). The cylinder contains a piston on a rod that moves a shutter in one of AccuRay's products. Compressed air moves the piston to open the shutter, and a spring in the cylinder, acting on the piston in the opposite direction to the air pressure, closes it. The piston moves in a sealed tube which is closed at one end by the cap, and at the other by the head. The rod passes through the Head. There are about 17 parts in all, some of them "off-the-shelf", but most are manufactured at AccuRay according to their design specifications.

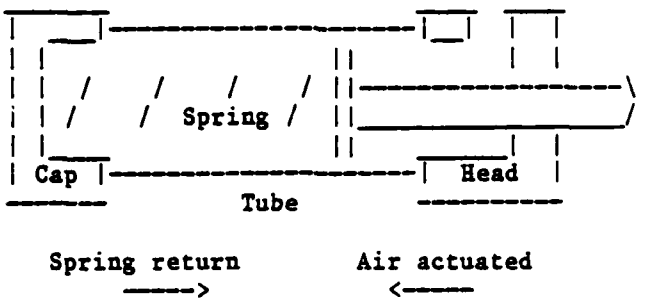


Figure 3 : Rough structure of Air-cylinder

After an extended set of interviews with the designer we have captured the 'trace' of the design process in some detail, and we are still refining it. The trace is the design decisions and their groupings considered over time. We have also isolated from this trace, in rough form, the conceptual structure and the plans that we consider underlie the design refinement process. As the implementation of the expert system progresses we expect the conceptual structures and the plans to become better defined.

7. Theoretical & Practical Issues

7.1 The language of design plans & inter-specialist communication

So far in our descriptions, plans are very general procedures. However, in order for this notion to have practical consequences in CAD we need generic representations for plans and the specification of their coordination. Otherwise, updating the expert system to reflect changing products or an increase in expertise will not be a practical. Thus, we need to search for planning primitives with which a language for design plans can be constituted. The issue of a plan specification and refinement language is especially important in our framework of problem solving types and corresponding specialist structures. We have successfully completed the task of specifying a language called CSRL for the specification of diagnostic specialists. We expect that a similar language can be designed to specify design specialists. We feel that earlier AI work on plans, such as that of Sacerdoti, Hayes-Roth and Bruce [6,7,8], will be applicable to our goal.

In our research, it is not the time sequencing of operations that is at issue, as plans have already been formed. We are concerned with the notion of constraint propagation from plans to subplans, either directly or via some blackboard [12]. However, each plan must show the sequence of tasks within that plan, some of which will use the expertise of other specialists to complete the task, and some of which will use a "compiled" [4] procedure to complete the task. Some specialists will be able to proceed in parallel.

Intimately bound up with the language of design plans is the nature of the communication between specialists. Most communication will be between a specialist and subspecialists (ie. the ones at the next lower level of the hierarchy, from which it is able to request action). The specialists may be asked to design or redesign, and could be asked to validate some small change that might affect them. Each message type will have some information associated with it. For example, the message requesting the redesign discussed above will require some information about the cause of the other specialist's failure, and, possibly, some suggestions from that specialist, or a "boss", about how to proceed.

7.2 How to handle failure

We have only scratched the surface of how failures of refinement result in reinvocation of portions of different plans. The issue is significantly more complicated. Sometimes when a plan failure occurs, it may be more beneficial to ask the parent specialist for some possibly minor changes in specifications rather than invoke *alternative plans*. As another example, consider the case where Plan 1 of a specialist S is being refined, and previous experience shows that a potential conflict in a specialist S' several levels below may be a most likely cause for failure of the plan. Let us also assume that several successors of S also have substantial responsibilities in the refinement of Plan 1 of S. Now it would seem prudent to selectively refine in the direction of S' to make sure early on that Plan 1 has a good chance of survival rather than engage all the relevant successors of S immediately. Finally, an important problem is how reasons for failure will be used by higher level specialists to choose alternate plans. Some degree of "understanding" the cause of failure will be necessary. At the very least some sort of classification of the causes of failure into categories that can be mapped into criteria for the selection of alternate plans will be necessary.

The above examples indicate that coordination of plans may become quite complex. Further research is called for concerning the trade-offs between overly complex plans that may capture some minor detail of the design process and sticking with

simpler plans that capture the essence of the design process, but perhaps lose some efficiency due to their incompleteness.

8. Discussion

Due to space limitations, we have not addressed several issues of interest, e.g., the conceptually important but common technique of "rough design" followed by a more detailed design based on some of the knowledge gained during the rough design phase, and the practically important problem of how to incorporate manufacturability constraints in the design process. Further it is likely that only a subset of practical industrial Class 3 problems can be successfully conquered by the design refinement paradigm. For some design tasks, we may have an insufficient understanding of the problem solving processes, or have difficulties with the amount of knowledge required. Nevertheless it is our belief that there is a significant subset of Class 3 design problems that are amenable to the proposed approach. The approach itself we think reflects in a natural manner the formation of conceptual structures for problem solving. Finally, it ought to be pointed out that while we have been mostly discussing the prospect of "automation" of design, the approach is also highly suited to semi-automation. An interactive system, in which the system, when faced with subtle issues concerning causes of failures of some designs, seeks human intervention at appropriate points in the plan selection process, will obviously be very useful. The knowledge decomposition principles that underlie our approach make the design of such semi-automatic systems particularly promising. When knowledge is decomposed into specialists, there is no particular constraint regarding which specialists need to be machine-implemented, and which can be given to human specialists.

9. References

[1]. Gomez, F., and Chandrasekaran, B., "Knowledge Organization and Distribution for Medical Diagnosis", IEEE Trans. Syst. Man. Cybern., SMC-11, 1981, pp. 34-42.

[2]. Mittal, S., and Chandrasekaran, B., "A Conceptual Representation of Patient Databases", Journal of Medical Systems, 4:2, 1980, pp. 169-185.

[3]. Chandrasekaran, B., "Towards a Taxonomy of Problem Solving Types", The AI Magazine, AAAI, Vol.4, No.1, pp.9-17, 1983.

[4] Chandrasekaran, B., and Mittal, S., "Deep Versus Compiled Knowledge Approaches to Diagnosic Problem-Solving", in Proc. Second National AI Conference, AAAI, Pittsburgh, Pennsylvania, 1982.

[5]. Fischer, G., and Bocker, H-D., "The Nature of Design Processes and How Computer Systems Can Support Them", European Conf. on Integrated Interactive Computive Systems, Stresa, Italy, Sept. 1982.

[6]. Sacerdoti, E., A Structure for Plans and Behavior, Elsevier, New York, 1977.

[7]. Hayes-Roth, B., Human Planning Processes, Rpt.No. R-2670-ONR, Rand Corp., Santa Monica, Calif., 1980.

[8]. Bruce, B., and Newman, D., Interacting Plans, Cognitive Science 2, p.195, 1978.

[9] Grinberg, M.R., A knowledge based design system for digital electronics, Proc. 1st Ann. Nat. Conf. on AI, p.283, Aug. 1980.

[10] Stefik, M. & Conway, L., Towards the principled engineering of knowledge, The AI Magazine, AAAI, Vol.3, No.3, p.4, 1982.

[11] Mitchell, T. et al, Representations for reasoning about digital circuits, Proc. 7th IJCAI, p.343, Aug 1981.

[12] Erman, L.D., A multi-level organization for problem-solving using many diverse cooperating sources of knowledge, Proc. 4th IJCAI, p.483, 1975.

[13] Latombe, J-C., (Ed), Artificial Intelligence and Pattern Recognition in CAD, North-Holland, 1978.

Acknowledgement: This work was supported by AFOSR grant #82-0255. We would also like to acknowledge the cooperation of the AccuRay Corporation and Pete Schmitz.

To appear in the Proc. Intern'l. Joint Conf.
on Artificial Intelligence, 1983.

CSRL: A LANGUAGE FOR EXPERT SYSTEMS FOR DIAGNOSIS

Tom Bylander, Sanjay Mittal*, and B. Chandrasekaran
Artificial Intelligence Group
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210 USA

Abstract

We present CSRL (Conceptual Structures Representation Language) as a language to facilitate the development of expert diagnosis systems based on a paradigm of "cooperating diagnostic specialists." MDX, the medical diagnosis system that has been developed in our laboratory over the past few years is based on this paradigm. In our approach, diagnostic reasoning is one of several generic tasks, each of which calls for a particular organizational and problem solving structure. A diagnostic structure is composed of a collection of specialists, each of which corresponds to a node or "concept" in a diagnostic hierarchy, e.g., a classification of diseases. A top-down strategy called establish-refine is used in which either a specialist establishes and then refines itself, or the specialist rejects itself, pruning the hierarchy that it heads. CSRL is a language for representing the concepts of a diagnostic hierarchy and for implementing the establish-refine process. The body of a concept specifies how it will respond to different messages from its superconcept. The knowledge to establish or reject a concept is factored into knowledge groups, which correspond to specific decisions in the diagnosis. We also introduce the concept of a family of languages in which different languages for diagnosis are designed for different kinds of end users.

I Introduction

Many kinds of problem solving for expert systems have been proposed within the AI community. Whatever the approach, there is a need to acquire the knowledge in a given domain and implement it in the spirit of the problem solving paradigm. Reducing the time to implement a system usually involves the creation of a high level language which reflects the intended method of problem solving. For example, EMYCIN was created for building systems based on MYCIN-like problem solving. Such languages are also intended to speed up the knowledge acquisition process by allowing domain experts to input knowledge in a form close to their conceptual level. Another goal is to make it easier to enforce consistency between the expert's knowledge and its implementation. In this paper, we present CSRL (Conceptual Structures Representation Language) as a language to facilitate the development of expert diagnosis systems based on the MDX approach to diagnostic problem solving [4, 8], an approach that has been developed in our laboratory over the past few years. In addition, we introduce the concept of a family of languages in which different languages are designed for different kinds of end users.

First, we will overview the relationship of MDX to our overall theory of problem solving types, the diagnostic problem solving that underlies MDX, and the differences between our approach and the knowledge base/inference engine approach. We then present CSRL in relationship to diagnosis and illustrate many of its constructs. Next, we discuss the family of languages concept. Finally, our immediate plans for using CSRL are listed. Due to space limitations, some understanding of how MDX performs diagnosis is assumed.

II Overview of MDX

A. Types of Problem Solving

Our group at Ohio State has been concerned with how knowledge is organized for expert problem solving. We propose that there are well-defined generic tasks each of which calls for a particular organizational and problem solving structure [3]. Some tasks that we have identified are diagnosis, consequence finding, and knowledge-directed data retrieval. The knowledge of a given domain that applies to a given task can be compiled into a knowledge structure which is tuned for that task. This structure is composed of a collection of specialists, each of which perform the same problem solving, but specialize in different concepts of the domain. Also, each task is associated with a problem solving regime, i.e., how the specialists coordinate for problem solving. The implementation of MDX is based on the diagnostic task.

B. The Diagnostic Task

The diagnostic task is the identification of a case description with a specific node in pre-determined diagnostic hierarchy. The idea of a diagnostic hierarchy is well-established in medicine in the form of disease classification.

*Currently at Knowledge Systems Area, Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA 94304 USA

(We will use medical terminology in the following, but the reader should keep in mind that the diagnostic task also applies to other domains, e.g., cars, computers, and power plants.) For example, figure 1 shows that cholestasis, cirrosis, and hepatitis are subclasses of liver disease. Cholestasis can be further refined into extra-hepatic and intra-hepatic cholestasis. In the diagnostic task, each disease is associated with a specialist that evaluates its presence or absence in a patient. Specialists in MDX, for example, attempt to classify a cholestatic case according to its etiology.

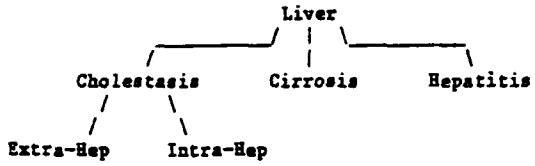


Figure 1: Fragment of a diagnostic hierarchy

A top-down strategy, which we call establish-refine, is used for this task. In relation to figure 1, a simple version of this strategy follows. First the Liver specialist determines if it is established, i.e., if liver disease is likely. If so, Liver refines itself by invoking its subspecialists. Each succeeding level of specialists performs the same establish and refine functions. On the other hand, if the Liver specialist rejects itself, the whole hierarchy of liver diseases can be pruned. This strategy, in combination with the diagnostic hierarchy, is the problem solving regime of the diagnostic task. For a detailed analysis of diagnostic problem solving, see Gomez and Chandrasekaran [7].

An important companion to the diagnostic hierarchy is a data base assistant which organizes the findings in a relevant manner [8, 9]. For example, to determine if a patient has been exposed to anesthetics, the data base, if necessary, can infer this from other data, e.g., major surgery or exposure to ether. Thus the diagnostic structure is insulated from solving problems about finding-finding relationships, avoiding a potentially combinatorial explosion of finding-disease relationships in the specialists of the diagnostic structure.

C. Differences

The usual approach to building knowledge based systems is to emphasize a general knowledge representation structure and different problem solvers which use that knowledge. One difference in the MDX approach is that the organization of its knowledge is not intended as a general representation for all problems. Rather it is tuned specifically for diagnosis. By limiting the type of problem to be solved, a specific organizational technique (classification hierarchy) and problem solving strategy (establish-refine) can be used to provide focus and control in the problem solving process.

Another difference is that the specialists in the hierarchy are not a static collection of knowledge. The knowledge of how to establish or reject is embedded within the specialists. Each specialist can then be viewed as a individual problem solver with its own knowledge base. The entire collection of specialists engages in distributed problem-solving.

III CSRL

CSRL is a language for defining a diagnostic hierarchy and for implementing the establish-refine process. A diagnostic hierarchy is represented by defining concepts. Relationships to neighboring concepts are specified in the declarations of the concept. Establish-refine is implemented within CSRL via message passing. Each concept has a body which specifies how the concept will respond to different messages, and which contains the statements which invoke other concepts with messages. The knowledge to establish or reject a concept is factored into knowledge groups, which determine how the case description relates to specific decisions in the diagnosis. For a complete description of CSRL, see Bylander [2].

A. Body and Message Blocks of a Concept

The body of a concept contains a list of message blocks, which specify how the concept will respond to different messages from its superconcept. The message block contains a message pattern, which is matched against the incoming message, and a sequence of CSRL statements, which are executed if the match succeeds. In figure 2, the body of Cholestasis contains two message blocks. The first one will be activated if an "Establish Cholestasis" message is sent from its superconcept, Liver (declared in the Declarations section), and the second, for a "Refine Cholestasis" message. The literal "Self" is bound to the name of the concept.

```
(Define-Concept Cholestasis
    (Declarations (Subconcept-of Liver)
                  ...)
    (Knowledge-Groups ...)
    (Body
        (Message-Block (Establish Self)
            ...)
        (Message-Block (Refine Self)
            ...)))
```

Figure 2: Message blocks in Cholestasis

Message blocks for establish messages are relatively simple since the knowledge groups (described below) do most of the work. Figure 3

shows how one would look for the Stone concept.* The knowledge groups are named Xray, Physical, History, and Summary. Within the (Establish Self) message block, an Execute statement runs all the knowledge groups, and then an Establish-Reply statement asserts the value of Summary as the establish value of Stone. The establish value is an integer from -3 to 3, which represents symbolic probabilities from "definitely not" to "definite." A value of 2 or 3 means that the concept has been established. This value is written on a blackboard [6], which other concepts can access.

```
(Define-Concept Stone
    (Declarations (Subconcept-of Extra-Hep)
                  ...)
    (Knowledge-Groups
        (Xray ...)
        (History ...)
        (Physical ...)
        (Summary ...))
    (Body
        (Message-Block (Establish Self)
            (Execute Xray History
                     Physical Summary)
            (Establish-Reply Summary))))
```

Figure 3: Statements for establishing Stone

Refining a concept is more complicated since the message block must be carefully tailored to follow the establish-refine strategy. In figure 4, the (Refine Self) message block contains two Callexpert statements. The first one calls each subconcept with an establish message (Subconcepts is bound to the declared list of subconcepts). The second Callexpert statement calls each subconcept that was established with a refine message.

Message passing is appropriate for the diagnostic task since the establish-refine regime easily translates into a message protocol, in which the messages clearly indicate the important activities of the concept. Also note that although each concept would have an establish message block in this formulation, the way that a concept establishes itself is concept-specific, i.e., a concept has its own knowledge groups.

B. Knowledge Groups

The Knowledge-Groups section contains a list of knowledge groups, which are used to evaluate how the case description relates to the establish value of a concept. A knowledge group (kg) can be thought of as a cluster of production rules which map the values of a list of conditions (boolean and arithmetic operations on data) to some conclusion on a discrete, symbolic scale. Different types of kg's perform this mapping differently, e.g.,

*Stone is a subconcept of Extra-Hep in MDX. It represents the disease "stone causing extra-hepatic cholestasis."

```
(Define-Concept Liver
    (Declarations (Subconcepts Cholestasis
                               Cirrosis
                               Hepatitis)
                  ...)
    (Knowledge-Groups ...)
    (Body
        (Message-Block (Refine Self)
            (Callexpert (E in Subconcepts)
                (With-Message (Establish E)))
            (Callexpert (E in Subconcepts)
                (With-Message
                    (Cond ((Established? E)
                           (Refine E))))))
        ...))
```

Figure 4: Statements for refining Liver

directly mapping values to conclusion, or having each rule add or subtract a set number of "confidence" units. Generally, the knowledge in a concept is factored into several kg's, and other kg's are used to combine their results. See [5] for a discussion on combining diagnostic knowledge in this way, as well as reasoning with uncertain data.

As an example, figure 5 is the Physical kg of the Stone concept presented above. The conditions query the data base (not defined in CSRL) for whether the patient has cholangitis, colicky pain in the liver, or has been vomiting. Each rule in the Match section is evaluated until one "matches." The value corresponding to this rule becomes the value of the kg. For example, the first rule tests whether the first and second conditions are true (the "?" means doesn't matter). If so, then 3 becomes the value of the knowledge group. Otherwise, other rules are evaluated. The resulting value of the table measures the strength of physical evidence towards establishing the Stone concept. The Xray and History kg's of Stone similarly evaluate the radiological and historical evidence. The Summary kg combines their results (the values of the other kg's are the conditions of Summary) into the establish value of Stone.

```
(Physical
   (Options (End-After (Match 1)))
   (Table (Conditions (Present? Cholangitis)
                      (Pain? Abdomen Colicky)
                      (Present? Vomit))
          (Match (If (T T ?) Then 3)
                 (If (? T T) Then 2)
                 (If (? T ?) Then 1)
                 (If (T ? ?) Then 1)
                 (If (? ? ?) Then -1))))
```

Figure 5: Example of a knowledge group

Factoring the knowledge of a concept in this manner has many advantages. Only the relevant knowledge gets invoked. It allows knowledge to be acquired more easily from domain experts because you can focus their attention on some specific

subtask. It also allows knowledge to be debugged because it is easier to see what purpose is being served by a knowledge group. This factoring would make it easier for experts to directly enter the knowledge at some future time.

C. Implementation of CSRL

CSRL is implemented on a DEC 20/60 using ELISP, a dialect of LISP developed at Rutgers, and a local version of FRL (Frame Representation Language). The CSRL interpreter and environment takes up an additional 33K words of storage. The environment includes a thorough syntax check when concepts are defined, commands to invoke any concept with any message, and a simple trace facility. CSRL currently allows little user interaction while it is running, but in the future we plan to add a simple explanation facility and to allow the user to "advise" the system during execution.

IV Family of Languages

Designing languages for knowledge representation often has to face conflicting requirements. At one end, they should be powerful enough to allow different kinds of knowledge and control to be expressed. The power is needed in the form of flexibility in the programming constructs available. At the other end, the language should be simple enough so that non-programmers such as domain experts can directly encode their knowledge without having to worry about the representation in the machine.

We are studying how to do this for the diagnostic task by using CSRL to experiment with the notion of "family of languages." The basic idea is that the same task is embedded in all languages in the family. However, some of the languages make stronger commitments to a particular message passing protocol or structuring of knowledge. Thus at the lowest level we have message passing and knowledge grouping but no commitment to any set of messages or any types of knowledge groups. In this regard, the language would become a general-purpose language such as LOOPS [1]. In fact, we are considering using LOOPS as the bottom-level of the diagnosis family.

The higher-level languages in the family would begin to tie these general facilities to the specifics of the diagnostic task. For example, a fixed set of message types may be allowed to carry out the message passing protocol of MDX. The highest levels may go so far as to create types of concepts, with built in templates for the knowledge groups and body. This would allow users to pick out the appropriate template and concentrate only on filling in the knowledge.

CSRL fits into this framework in the following way. A strong commitment is made concerning the types of knowledge groups that are available, but no commitment is made as to the set of messages that must be used. However, the flow of control is definitely restricted to be top-down.

V Current Plans

Our group at Ohio State is currently using CSRL in a variety of domains including blood type analysis, cars, and nuclear power plants. We are also translating MDX's diagnostic structure from the present LISP code to CSRL. We also plan to implement a diagnosis language which non-programmers can use with minimal training to implement prototype diagnostic systems.

Acknowledgments

We would like to acknowledge Jack Smith and Jon Sticklen for several discussions during CSRL's design phase. The language development is funded by a grant from the Battelle Memorial Laboratories University Distribution Program, and experimentation and application in different domains is supported by AFOSR grant 82-0255, and NSF grant MCS-8103480.

References

1 D. Bobrow and M. Stefik, "The LOOPS Manual," Tech. Report KB-VLSI-81-13, Xerox Palo Alto Research Center, 1982.

2 T. Bylander, "The CSRL Manual," in preparation, 1983.

3 B. Chandrasekaran, "Towards a Taxonomy of Problem Solving Types," AI Magazine, Vol. 4, No. 1, Winter/Spring 1983.

4 B. Chandrasekaran, F. Gomez, S. Mittal, and J. W. Smith, "An Approach to Medical Diagnosis Based on Conceptual Structures," in Proc. IJCAI-79, 1979.

5 B. Chandrasekaran, S. Mittal, and J. W. Smith, "Reasoning with Uncertain Knowledge: The MDX Approach," in Proc. 1st Ann. Joint Conf. of the American Medical Informatics Association, May 1982.

6 L. D. Erman and V. R. Lesser, "A Multi-level Organization for Problem-solving using Many Diverse Cooperating Sources of Knowledge," in Proc. IJCAI-75, 1975.

7 F. Gomez and B. Chandrasekaran, "Knowledge Organization and Distribution for Medical Diagnosis," IEEE Trans. SM&C, Vol. SMC-11, No. 1, pp. 34-42, January 1981.

8 S. Mittal, "Design of a Distributed Medical Diagnosis and Database System," Ph.D. Thesis, Department of Computer and Information Science, The Ohio State University, 1980.

9 S. Mittal and B. Chandrasekaran, "Conceptual Representation of Patient Data Bases," J. of Medical Systems, 1981.

APPLICATION OF THE CSRL LANGUAGE TO THE DESIGN OF EXPERT DIAGNOSIS SYSTEMS: THE AUTO-MECH EXPERIENCE

by

Michael C. Tanner and Tom Bylander

Department of Computer and Information Science

The Ohio State University

Columbus, Ohio, USA 43210

November 12, 1983

Table of Contents

List of Figures

Abstract

Auto-Mech is an expert system which diagnoses automobile fuel systems. Its organization and strategies are patterned after MDX, an expert diagnosis system developed in our AI laboratory. The problems that these systems are able to diagnose are represented as nodes within a hierarchy. Each node has knowledge about how to confirm or reject the problem hypothesis, as well as knowledge about what nodes to consider next. This approach is intended to be a domain-independent methodology for providing focused problem solving and for localizing knowledge in a conceptually relevant manner. Auto-Mech is implemented in a recently developed language called CSRL, which is specifically intended for building diagnostic expert systems. This paper describes Auto-Mech and discusses why the MDX approach and CSRL were useful in developing Auto-Mech, and where some difficulties were encountered.

1. Introduction

Over the past few years, our AI laboratory has developed an approach to the design of expert diagnosis systems based on the paradigm of "cooperating specialists." This approach is exemplified in an expert system called MDX [3, 6], whose expertise is in cholestatic liver disease. In order to demonstrate the viability of this approach to non-medical domains, we have developed a system called Auto-Mech which diagnoses problems in automobile fuel systems. We show that an organization of diagnostic knowledge which is similar to MDX can be used in this domain to provide focused problem solving, and to localize knowledge in a conceptually relevant manner.

Auto-Mech is implemented in a recently developed language called CSRL [2], which was designed specifically for building MDX-like diagnostic expert systems. Thus another goal of this work was to determine the strengths and weaknesses of CSRL and to make recommendations for future versions of CSRL.

Briefly, Auto-Mech works as follows. When Auto-Mech begins diagnosis, it obtains a specific complaint about the way the car operates. Then general hypotheses about the nature of the problem are evaluated. When a hypothesis is confirmed, any hypotheses which are immediately more specific are considered. The user is queried for additional information as needed during

this process. Auto-Mech is not intended to be a complete model of an automobile mechanic, but is intended to reflect the information processing capability of a mechanic when she attempts to determine the specific cause of a fuel problem from an initial complaint and from things that a typical mechanic can observe when she looks under the hood.

Before we present a more detailed description of Auto-Mech, we give an overview of our approach to diagnostic problem solving. We then describe the program, explaining the assumptions that we have made, and outlining its organization. An annotated session of Auto-Mech and a sample of its CSRL code is included. Finally, we discuss why our approach and CSRL were useful in developing Auto-Mech, and where some difficulties were encountered.

2. Introduction to Diagnostic Problem Solving

The central problem solving of diagnosis, in our view, is classificatory activity. This is a specific type of problem solving in our approach, meaning that a special kind of organization and special strategies are strongly associated with performing expert diagnosis. We will not examine here how classificatory diagnosis fits in with our overall theory of problem solving (see Chandrasekaran [4]). Instead, we will briefly overview the structure and the strategies of classificatory diagnosis. For the purposes of this discussion, we will use "diagnosis" in place of "classificatory diagnosis" with the understanding that the complete diagnostic process includes other elements as well.

The diagnostic task is the identification of a case description with a specific node in a pre-determined diagnostic hierarchy. Each node in the hierarchy corresponds to a hypothesis about the state of the "patient" (a car in the Auto-Mech program). Nodes higher in the hierarchy represent more general hypothesis, while lower nodes are more specific. In medicine, a case description is the manifestations and the history of a patient, and a diagnostic hierarchy is a classification of diseases and disease classes. For example, MDX [3, 6] attempts to classify a medical case into a diagnostic hierarchy of cholestatic diseases. Figure 1 illustrates a fragment of MDX's hierarchy. The most general disease, cholestasis in this example, is the head

node of the hierarchy. More specific cholestatic diseases such as extra-hepatic cholestasis are classified within the hierarchy. In the following discussion, we will use the generic term "problem" rather than "disease".

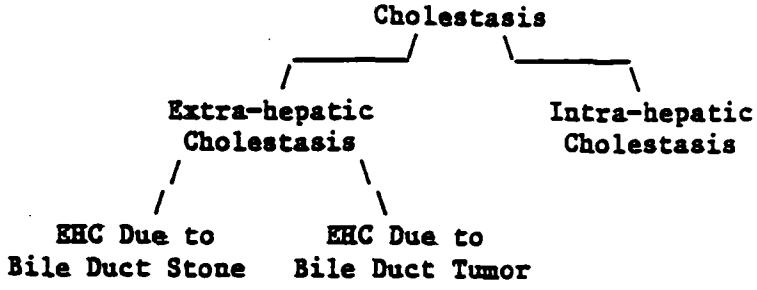


Figure 1: Fragment of MDX's diagnostic hierarchy

Each problem in the hierarchy is associated with a specialist which contains the diagnostic knowledge to evaluate the presence or absence of the problem from the case description. From this knowledge, the specialist determines a confidence value representing the amount of belief that the problem exists. If this value is high enough, the specialist is said to be established. Note that each specialist is a problem solver with its own knowledge base.

The basic strategy of the diagnostic task is a process of hypothesis refinement, which we call establish-refine. In this strategy, if a specialist establishes itself, then it refines the problem hypothesis by invoking its subspecialists, which also perform the establish-refine strategy. If the confidence value is low, the specialist rejects the problem hypothesis, and performs no further actions. Note that when this happens, the whole hierarchy below the specialist is eliminated from consideration. Otherwise the specialist suspends itself, and may later refine itself if its superior requests it. The processing ends (if we assume that only one problem is present) when a tip node specialist, a specialist with no subspecialists, has been established.

With regard to Figure 1, the following scenario might occur. First, the cholestasis specialist is invoked, since it is the top specialist in the hierarchy. Cholestasis is then established, and the two specialists below it are invoked. Extra-hepatic cholestasis is rejected, also eliminating EHC due

to stone and bile duct cancer from further consideration. Finally, intrahepatic cholestasis establishes itself, and invokes its subspecialists.

Due to space and time limitations, we have not addressed several issues relevant to diagnostic problem solving (such as handling multiple problems). For a more detailed analysis, see Gomez and Chandrasekaran [5]. Test ordering, causal explanation of findings, and therapeutic action do not directly fall within the auspices of classificatory diagnosis, but expertise in any of these areas would certainly enhance a diagnostic system. Fully resolving these issues and integrating their solutions into the diagnostic framework are problems for future research.

3. The Automobile Diagnosis Program

3.1. Description of Auto-Mech

Auto-Mech is a program which diagnoses fuel problems in automobile engines. It was developed using CSRL (which will be described in Section 3.3) and the establish-refine problem-solving methodology described in Section 2.

One reason the domain of automobile diagnosis was chosen is that most people feel comfortable discussing car problems thus making such a program easy to demonstrate. We also had two good amateur mechanics available to serve as experts. We decided to concentrate on fuel problems because the fuel system is sufficiently complex to be interesting and simple enough to do in a short time.

Before discussing the program further a brief discussion of automobile fuel systems is in order. The purpose of the fuel system is to deliver a mixture of fuel and air to the cylinders of the engine. It can be divided into four major subsystems:

1. the fuel delivery subsystem which brings fuel from the tank to the carburetor,
2. the air intake which brings air into the carburetor,
3. the carburetor which mixes the air and fuel in the proper ratio, and

4. the vacuum manifold which brings the mixture to the cylinders.

These subsystems correspond to initial hypotheses about fuel system faults and each can be further refined by more detailed descriptions.

Just as hospitals have a routine series of data to collect about every patient admitted, Auto-Mech collects a set of initial data to get the diagnosis running. We refer to the initial data as defining the user's complaint. The complaint is a problem-condition pair where the problem is the symptom the user notices (such as stalling or running rough) and the conditions include the kind of driving in which the problem occurs (accelerating, idling, etc.) and the approximate engine temperature (hot, cold, or both). Note that the complaint is highly symptomatic.

We chose to implement a program around a generic automobile fuel system rather than the fuel system of a particular car. Reasoning about the fuel system depends on its design, which can vary in many ways. Within the CSRL framework each design requires its own diagnostic hierarchy so we had to make a few assumptions about the system. The major assumptions are:

- carbureted engine
- single barrel, single stage, downdraft carburetor
- mechanical fuel pump
- automatic transmission
- non-computer ignition
- automatic choke
- minimal pollution control systems

Each of these assumptions has diagnostic consequences. A carbureted engine, for example, will have a different set of problems than a fuel injected engine (the former can have a broken carburetor). Many of these assumptions would be valid for most cars built before 1980 or so. Those that are not would either add complexity without making the problem more interesting (such as a two-stage carburetor) or vary so widely that no single generic arrangement can be imagined (such as pollution controls).

We also made a few simplifying assumptions about the problem solving required of the program. The most important of these is our single complaint assumption.* This means that for any session with the program the user can specify only one major complaint (a problem-condition pair as described above). One difficulty with multiple complaints is the need to keep the problem-condition pairs together. If the complaints were "stalls while idling" and "hesitates on acceleration" it would be necessary to know "stalls", "hesitates", "idling", "acceleration", and that the complaints are not "stalls on acceleration" and "hesitates while idling". The simple data base provided with CSRL does not provide for this kind of reasoning. This could be rectified by implementing a special data base. Another difficulty with allowing many complaints is keeping the line of questioning focused on one complaint. Given many complaints, large portions of the hierarchy will be relevant and the questioning may appear random to a user unless some mechanism is used for focusing the questioning. Such a mechanism was not readily available. Solving these problems would have either added to the time taken for the project as a whole or subtracted from the time devoted to the main purpose of developing Auto-Mech — to test CSRL and establish-refine problem-solving. So we chose to restrict the problem-solving to a single complaint at a time.

Another simplifying assumption we made is that the data to be used by the system be from commonly available sources. Mechanics now have an array of computer analysis information available which our experts were unfamiliar with. So we limited ourselves to such data as whether a component is working and how the car behaves in certain situations.

Figure 2 shows part of the diagnostic hierarchy for Auto-Mech. Each node in the hierarchy is a specialist representing a hypothesis together with knowledge about how to confirm or reject the hypothesis. For example, the specialist named Delivery represents the hypothesis "Fuel delivery subsystem is causing the problem." Delivery also contains knowledge about the types of

---

*This is most emphatically <u>not</u> a single fault assumption. If there is more than one fault causing the complaint, Auto-Mech can find it.

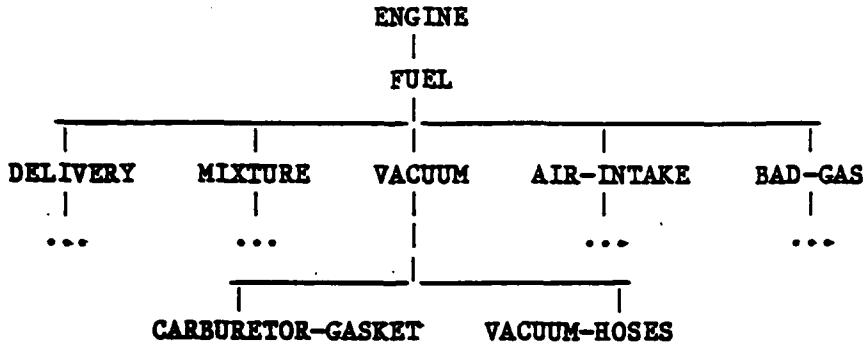


Figure 2: Partial Diagnostic Hierarchy for Auto-Mech

complaints for which fuel delivery problems should be considered and how to infer that fuel is not being delivered to the carburetor. The purpose of the top-level specialist, Engine, is to collect the initial complaint information and begin diagnosis. The ellipsis in the diagram represent points where the hierarchy continues down.

3.2. Annotated Transcript of a Session with Auto-Mech

In the following transcript of a session with Auto-Mech the user's responses follow the ">" prompt, comments are underlined, and everything else comes from the program.

> The user first tells the top level specialist, Engine, to establish and then refine itself. The program then prints a brief statement of the diagnostic system's scope and begins collecting information about the problem.

```
>DoCSRL Engine Establish-Refine

This is a program to diagnose automobile problems. At the present
time the the program only knows about fuel problems. The first series
of questions is an attempt to determine what the problem is (starting
problem, rough running, stalling, hesitation, slow response,
knocking), and the conditions under which it occurs (idling, loading,
accelerating, cruising, turning, decelerating, engine temperature).
After this the rest of the questions are attempts to confirm or reject
hypotheses.

Do you have problems starting your car?>?

  Answers:
     Y = Yes
     N = No
```

U = Unknown

If the user doesn't know how to respond, "?" will provide a description of acceptable answers. All of the questions in this program are answered "yes", "no", or "unknown" for simplicity. CSRL does provide facilities for using any answers which are appropriate including such things as numerical values and multiple choice.

Do you have problems starting your car?>N

Does the car stall?>N

Does the car run rough?>Y

Does the problem occur while idling?>N

Does the problem occur on loading?>Y

Does the problem occur while the engine is both hot and cold?>Y

The problem the user has specified is "the engine runs rough" and the conditions are "on loading and independent of engine temperature". The only purpose of the Engine specialist is to determine the complaint. Certain terms require specific definitions in order to clearly separate some problems and conditions. For example, loading means putting some strain on the engine without accelerating, idling means the engine is running but there is no load on it.

>>> Message Trace <<<
From: ENGINE To: FUEL
Message: ((ESTABLISH FUEL))

Engine now refines by first telling its subspecialist, Fuel, to establish.

Have you eliminated ignition as a possible cause of the problem?>Y

This question shows the need to know what other specialists have done. Auto experts have determined that many problems which might be fuel problems are more likely to be ignition problems. This user's auto complaint is one of those cases so Fuel wants to make sure Ignition has rejected itself. However, Ignition has not been implemented so the user is asked if the ignition system has been considered and rejected.

>>> Message Trace <<<
From: FUEL To: ENGINE
Message: ((ESTABLISHED FUEL 2))

```
>>> Message Trace <<<
From: ENGINE To: FUEL
Message: ((REFINE FUEL))
```

Once its subspecialist, Fuel, establishes, Engine continues to refine by telling Fuel to refine itself. The next series of messages and questions show the program considering Delivery, Mixture, Vacuum, Air-Intake, and Bad-Gas as hypotheses about the cause of the problem. The data base provided with CSRL records each question asked and the user's answers to avoid asking them again. So the decisions the specialists make are not based entirely on the answers to questions shown under each specialist, but on combinations of the answers to those and previous answers.

```
>>> Message Trace <<<
From: FUEL To: DELIVERY
Message: ((ESTABLISH DELIVERY))

Is any fuel delivered to the carburetor?>U

>>> Message Trace <<<
From: DELIVERY To: FUEL
Message: ((REJECTED DELIVERY -2))

>>> Message Trace <<<
From: FUEL To: MIXTURE
Message: ((ESTABLISH MIXTURE))

Have you been getting bad gas mileage?>N

>>> Message Trace <<<
From: MIXTURE To: FUEL
Message: ((REJECTED MIXTURE -3))

>>> Message Trace <<<
From: FUEL To: VACUUM
Message: ((ESTABLISH VACUUM))

>>> Message Trace <<<
From: VACUUM To: FUEL
Message: ((ESTABLISHED VACUUM 3))

>>> Message Trace <<<
From; FUEL To: AIR-INTAKE
Message: ((ESTABLISH AIR-INTAKE))

Is the air filter old?>N

>>> Message Trace <<<
From: AIR-INTAKE To: FUEL
Message: ((REJECTED AIR-INTAKE -2))
```

```
>>> Message Trace <<<
From: FUEL To: BAD-GAS
Message: ((ESTABLISH BAD-GAS))

Have you tried a higher grade of gas?>Y

>>> Message Trace <<<
From: BAD-GAS To: FUEL
Message: ((REJECTED BAD-GAS -3))

>>> Message Trace <<<
From: FUEL To: VACUUM
Message: ((REFINE VACUUM))
```

Fuel now asks its established subspecialists to refine. In this case only Vacuum has established.

```
>>> Message Trace <<<
From: VACUUM To: VACUUM-HOSES
Message: ((ESTABLISH VACUUM-HOSES))

Are there any cracked, punctured or loose vacuum hoses?>U
```

This question seems strange because it appears to be equivalent to asking whether the hypothesis should be confirmed. But when Auto-Mech gets to a very specific hypothesis usually the only data for confirming or rejecting it comes from direct observation of a part.

```
Can you hear hissing while the engine is running?>N

Are the vacuum hoses old?>Y

>>> Message Trace <<<
From: VACUUM-HOSES To: VACUUM
Message: ((UNKNOWN VACUUM-HOSES 1))
```

The information that Vacuum-Hoses has is not certain, some indicates trouble and some doesn't. So the answer is "unknown", but the value "1" indicates that it leans toward establishing.

```
>>> Message Trace <<<
From: VACUUM To: CARBURETOR-GASKET
Message: ((ESTABLISH CARBURETOR-GASKET))

Can you see cracks in the carburetor gasket?>Y

>>> Message Trace <<<
From: CARBURETOR-GASKET To: VACUUM
Message: ((ESTABLISHED CARBURETOR-GASKET 3))
```

Here is an example of how the hierarchy sets context for lower level specialists. The carburetor gasket often appears cracked or split but does not cause problems. Cracks in it are thus indicative of trouble only in a context in which a vacuum leak is suspected.

```
Is the diagnosis of VACUUM finished?>Y
```

CSRL and Auto-Mech are unable to determine when diagnosis is finished. The mechanism we use asks the user as control passes up through the hierarchy from the lowest point reached. If the user answers "Yes", as in this case, then control passes on up the hierarchy. Another of the user's options here is to answer "No". In that case CSRL would refine those subspecialists of Vacuum which were "unknown", such as Vacuum-Hoses. Unless the program is told to do this only "established" subspecialists will get refined. In this particular case the question indicates a bug in the Auto-Mech program itself since Vacuum's subspecialists are all tip specialists.

```
>>> Message Trace <<<
From: VACUUM To: FUEL
Message: ((ESTABLISHED CARBURETOR-GASKET 3) (UNKNOWN  VACUUM-HOSES 1))

Is the diagnosis of FUEL finished?>Tree

FUEL — 2
   DELIVERY — -2
   MIXTURE — -3
   VACUUM — 3
       VACUUM-HOSES — 1
       CARBURETOR-GASKET — 3
   AIR-INTAKE — -2
   BAD-GAS — -3
```

The user also has the option of printing out the diagnostic hierarchy with the values displayed for each specialist.

```
Is the diagnosis of FUEL finished?>Y

>>> Message Trace <<<
From: FUEL To: ENGINE
Message: ((UNKNOWN VACUUM-HOSES 1) (ESTABLISHED  CARBURETOR-GASKET 3)
(REJECTED BAD-GAS -3) (REJECTED  AIR-INTAKE -2) (ESTABLISHED VACUUM 3)
(REJECTED MIXTURE -3) (REJECTED DELIVERY -2))

Is the diagnosis of ENGINE finished?>Y

(ANSWER (REJECTED DELIVERY -2)
        (REJECTED MIXTURE -3)
```

```
(ESTABLISHED VACUUM 3)
(REJECTED AIR-INTAKE -2)
(REJECTED BAD-GAS -3)
(ESTABLISHED CARBURETOR-GASKET 3)
(UNKNOWN VACUUM-HOSES 1)
(ESTABLISHED FUEL 2)
(ESTABLISHED ENGINE 3))
```

The answer is simply a list of the specialists which ran and their values. The diagnosis is the established tip specialists, Carburetor-Gasket in this case.

3.3. How One of Auto-Mech's Specialists Reasons

Figure 3 shows the CSRL code for implementing the Bad-Gas specialist which considers the hypothesis "Something wrong with the fuel is causing the problem." The specialist is defined by the Define-Concept statement. Like all CSRL specialists it is made of three parts:

- Declarations, containing information about where the specialist fits in the hierarchy.
- Knowledge-Groups, showing the major categories of decisions to be made.
- Body, which controls the way in which the specialist responds to various messages.

The boldface represents built-in CSRL primitives, everything else is determined by the system builder. And-YNU is a three-valued logical AND which is defined for Y, N, and U. Use-Declaration and Use-Statement invoke CSRL macro-instructions that expand into longer sequences of statements which do not vary from specialist to specialist. The Use-Declaration's here set up standard variables and constants for the CSRL interpreter to use. The Use-Statement's implement the establish-refine problem-solving process. Since the interesting thing is how Bad-Gas establishes or rejects itself, we will not discuss these other processes here.

The general description of how Bad-Gas reasons is:

> First make sure Bad-Gas is a relevant hypothesis to hold. If it is not then reject. If it is relevant find out if there is any reason to believe something has happened to the fuel recently. If there is none then reject. But if there is some reason to believe this then establish with value depending on how relevant the hypothesis is.

```
(Define-Concept Bad-Gas
  (Declarations (Subconcept-Of Fuel)
                (Subconcepts Low-Octane
                             Water-In-Fuel
                             Dirt-In-Fuel)
                (Use-Declaration Usual-Variables)
                (Use-Declaration Usual-Constants))
  (Knowledge-Groups
     (Relevant
        (Options (End-After (Match 1)))
        (Table (Conditions
                (Ask-YNU? "Is the car slow to respond")
                (Ask-YNU? "Does the car start hard")
                (And-YNU
                 (Ask-YNU? "Do you hear knocking or pinging sounds")
                 (Ask-YNU? "Does the problem occur while accelerating")))
               (Match (If ( Y  ?  ? ) Then -3)
                      (If ( ?  Y  ? ) Then -3)
                      (If ( ?  ?  Y ) Then  3)
                      (If ( ?  ?  ? ) Then  1))))
     (Gas
        (Options (End-After (Match 1)))
        (Table (Conditions
                (Ask-YNU? "Have you tried a higher grade of gas")
                (Ask-YNU? "Did the problem start after the last fillup")
                (Ask-YNU? "Has the problem gotten worse since the last
                           fillup"))
               (Match (If ( Y  ?  ? ) Then -3)
                      (If ( ?  Y  ? ) Then  3)
                      (If ( ?  N  Y ) Then  2)
                      (If ( ?  ?  ? ) Then -3))))
     (Summary
        (Options (End-After (Match 1)))
        (Table (Conditions Relevant  Gas)
               (Match (If (   3     (Ge 0) ) Then  3)
                      (If (   1     (Ge 0) ) Then  2)
                      (If (   ?     (Lt 0) ) Then -3)))))
  (Body
     (Use-Statement Usual-Establish-Refine)
     (Message-Block (Establish Self)
         (Execute Relevant)
         (Case Relevant
           ((Ge 0)(Execute Gas Summary)
                  (Establish-Reply Summary))
           (Otherwise (Establish-Reply Relevant))))
     (Use-Statement Simple-Refine)
     (Use-Statement Pass-Messages)))
```

Figure 3: CSRL code for a specialist

To implement this Bad-Gas has a group of statements in the Body which begin

```
(Message-Block (Establish Self) ... )
```

and which will be activated when a message to establish is received from the Fuel specialist.* The Message-Block controls the order in which the Knowledge-Groups (Relevant, Gas, and Summary) are evaluated. Summary combines the results of Relevant and Gas. The

```
(Execute Relevant)
```

statement causes the Relevant knowledge-group to run. If Relevant returns a non-negative value the Gas and Summary groups run with the establish-value of Bad-Gas set by Summary. If Relevant returns a negative value the establish-value of Bad-Gas is set by Relevant and the other two groups are not run. This choice is implemented by the construct:

```
(Case Relevant
  ((Ge 0)     ... )
  (Otherwise ... ))
```

Very detailed descriptions of how all of the knowledge groups work is not necessary. In general, running a knowledge-group consists of testing its Conditions and trying to match their results to one of the rows in the Match table. A condition which begins with Ask-YNU? causes CSRL to look in its string-value data base for the given string. If found then the value stored there becomes the value of the condition. If not found, the string is displayed as a question to the user. The user's response is stored in the string-value data base and is used as the value of the condition. If the condition is the name of a knowledge-group its value is the value of the knowledge-group. The rows (or "rules") of the Match table are tried one at a time, from the top down. As soon as a row is found which matches the value of the conditions, the Then-part gives the value of the knowledge-group and the evaluation of the knowledge-group stops. The "?" in the tables is a wild-

---

*CSRL can be viewed as a restricted object oriented language in which the objects are the specialists and the messages are instructions to the specialists.

card, it matches any value.

In the transcript given earlier the value of the conditions in the Relevant knowledge-group are (N N N) so the row

If (? ? ?) Then 1

matches and the value of Relevant is 1. This is a result of the previously supplied information that the complaint is "runs rough on loading" combined with the single complaint assumption. As a result Gas and Summary are run. The value of the conditions in Gas are (Y - -), where "-" signifies "did not ask", and the row

If (Y ? ?) Then -3

matches. This is the result of asking a question of the user. So now the values of the conditions for Summary are (1 -3) which matches

If (? (Lt 0)) Then -3

and the value of Summary is -3, causing Bad-Gas to reject.

For more detail about CSRL see [2] and [1].

3.4. Usefulness of CSRL in Developing Auto-Mech

One of the first things we noticed in using CSRL is that the internal workings of a specialist and the overall problem-solving method is easy to explain to a computer-naive expert.* Establish-refine seemed to be a natural way for the experts to solve the problems and was not in any way imposed upon them. The specialists in the diagnostic hierarchy of Auto-Mech represent the hypotheses considered by the experts during the solution of practice problems. The experts quickly understood the CSRL specialists and could point out flaws in their reasoning during debugging sessions.

---

*Our experts were Ph.D. students in Nuclear Engineering, they were not computer specialists and knew very little about AI.

Another helpful feature of CSRL is that it makes it easy to get something running quickly. This gives the experts a chance to actually run the program to see the results of their suggestions. It is much easier for experts to help debug a running program than to debug a paper construct. CSRL makes possible the development of partial systems, the obvious evidence for which is that we can develop a fuel system program without being concerned with the rest of the car. Much of this is due to our approach to diagnosis in which knowledge is localized within specialists and the interaction among specialists is simple and well-defined. Concerns about global interaction of knowledge are minimized. Changes in the Delivery specialist will not affect any other specialists in the hierarchy (except for the context assumed by its subspecialists, an easy thing to check). So if Vacuum works right but Delivery has bugs in it, fixing Delivery will not affect Vacuum. This greatly simplifies building and debugging a system over the traditional knowledge-base/inference-engine approach.

Auto-Mech consists of 34 specialists in a hierarchy which varies from four to six levels deep. Four people were actively involved with its development, two computer specialists and two domain experts. The total labor was approximately five man-months of which about 30% was domain expert time. The project extended over nine calendar months.

3.5. Some Difficulties

CSRL was built to embody a theory of diagnosis which was developed in the medical domain. The diagnostic reasoning of an automobile mechanic, however, is slightly different from that of a doctor. Once a hypothesis is confirmed a doctor will carefully consider the competing refinement hypotheses and follow up on the best. This is the behavior modeled by the establish-refine theory. But an auto mechanic seems to follow up the first reasonable refinement. Auto-Mech does not capture this latter behavior. It could be done in CSRL, though it would be a little more difficult to do than using the standard establish-refine routines. The end result of diagnosis in both cases is the same, but presently Auto-Mech seems dumb to an expert since it is being more careful than necessary.

Mechanics usually do not go straight into the kind of diagnostic reasoning which requires a diagnostic hierarchy. The complaints are associated with typical maintenance or repair procedures as a result of the training or experience of the mechanic. An example of this is:

> Temperature dependent problems (those that happen only when the engine is cold or only when it is hot) are usually caused by a malfunction of the choke. So for those problems first check to see that the choke works correctly and if it does not then fix it. Also, since you have to go past the air filter to get to the choke, make sure the air filter is good.

Only after all the applicable procedures like this get tried does the mechanic do the more serious diagnosis done by Auto-Mech. This process is outside the scope of both CSRL and the establish-refine theory but it is actually only an efficiency measure. The final diagnoses using the mechanic's method and using establish-refine are the same, with establish-refine possibly doing more work.

One of the problems we had in developing Auto-Mech is that we could not treat establishing a specialist, or confirming a hypothesis, as indicating a high degree of belief in the hypothesis. Sometimes in Auto-Mech a specialist establishes because it is not possible to reject it and one of its subspecialists may be able to establish. So during diagnosis, when a specialist establishes it really means "this hypothesis is worth pursuing." This is mainly due to the type of domain that we were working with. Most of the data used by Auto-Mech are very weak at indicating specific problems. Data that are direct are usually about tip hypotheses and of the form "is xxx working correctly." Thus the specialists which rely on indirect data are unable to produce high confidence in their associated hypotheses, although they can still determine a "pursuit value." The theory of establish-refine problem solving, which CSRL is based on, needs to be modified to take this into account.

The question of when to stop is a difficult one for diagnosis in general. Human experts have the concept of a diagnosis "explaining" the data and that certain data must be explained while other data need not be. Automating this decision has proven to be difficult, though it seems clear that it is not a decision appropriately made by the diagnostic expert itself but rather by some

outside entity having additional knowledge and skills. This is why CSRL currently asks if the user is satisfied as control passes back up through the hierarchy. However, in the automobile domain the system should recommend fixing the problems represented by established tip specialists and then ask if the problem persists after the repairs are made. The answer to that question could be data for another round of diagnosis. Once again, this could be fixed within CSRL (the problem arises from the built-in Simple-Refine macro which was designed to be very general and would need to be customized for Auto-Mech).

In the establish-refine theory of diagnostic problem-solving, diagnosis is seen as an inherently parallel process [5]. All specialists at a given level in the hierarchy may be active simultaneously. These specialists communicate their status (established or rejected) to each other via a blackboard. This makes it possible for a specialist to know what another specialist has done. Sometimes this knowledge is necessary, as can be seen in the example session given earlier where the Fuel specialist wanted to know the status of the Ignition specialist. CSRL is presently implemented as a serial language without a blackboard. This leads to some occasional awkwardness as the Fuel specialist's question to the user shows.

Another feature of establish-refine theory is that it is strictly a theory of classificatory reasoning and that other kinds of reasoning are needed to do diagnosis. In particular inferential reasoning about data is needed. For example, if the user's problem is one which involves the engine running then the system should know that there is fuel in the tank even if that piece of data is not explicitly given to it. This is reasoning about relationships between pieces of data and is not classificatory in nature. In MDX there is an intelligent data base component, called PATREC [6, 7], for doing such reasoning about medical data. CSRL is intended to be used for the diagnostic component and so it does not contain an intelligent data base. Since this component is absent in Auto-Mech, we have had to clutter our diagnostic specialists with data base reasoning.

4. Summary and Recommendations

The relative length of the "difficulties" section compared to the "usefulness" section is due to the need for additional programming and the CSRL language rather than deficiencies in the theory of diagnostic problem-solving. The difficulties point to some recommendations for improvements in CSRL:

1. Changes from the standard control flow should be easier to make. Presently all hypotheses at one level are tried before going down to the next level. The control needed by Auto-Mech is a natural complement to this — pursue the first reasonable hypothesis.

2. CSRL needs a more flexible facility for determining when to stop than simply asking the user. System builders may have ideas on how to do it for specific problems without necessarily being able to solve the general problem of deciding when diagnosis is complete.

3. Since CSRL is to be used for building diagnostic experts it should provide a facility for limited communication between specialists across the hierarchy, such as a blackboard. Such a facility would be useful even if CSRL remains a serial language.

Our other problems were the result of things which it would not be appropriate for CSRL to address since they are outside the scope of classificatory diagnosis. These include the intelligent data base and the execution of typical maintenance procedures prior to diagnosis.

Overall CSRL was a very useful tool for developing a diagnostic expert system. It was easy to explain to an expert, the specialists were fairly easy to write based on protocols, and a partial system could be running quickly for debugging purposes. CSRL was also easy to use from a programmer's point of view.

Auto-Mech does not verify the validity of establish-refine problem-solving but it does demonstrate that establish-refine is a viable method for doing diagnosis. It is a natural way for experts to solve problems. The hypotheses they consider can be used as specialists within the diagnostic system. The localization of knowledge proved to be useful for development purposes.

Acknowledgments

We thank B. Chandrasekaran for his guidance and support during the course of this project. We also wish to thank Jiten Ruparel and Siavash Hashemi who served as our domain experts and Sriram Mahalingam who helped think out the problems associated with building Auto-Mech. Research on diagnostic expert systems for the domain of mechanical systems is supported by the Air Force Office of Scientific Research grant 82-0255.

References

1. T. Bylander. The CSRL Manual. in prep.

2. T. Bylander, S. Mittal, and B. Chandrasekaran. CSRL: A Language for Expert Systems for Diagnosis. IJCAI-83, 1983.

3. B. Chandrasekaran, F. Gomez, S. Mittal, and J. W. Smith. An Approach to Medical Diagnosis Based on Conceptual Structures. IJCAI-79, 1979.

4. B. Chandrasekaran. "Towards a Taxonomy of Problem Solving Types." AI Magazine 4, 1 (Winter/Spring 1983), 9-17.

5. F. Gomez and B. Chandrasekaran. "Knowledge Organization and Distribution for Medical Diagnosis." IEEE Trans. SM&C SMC-11, 1 (January 1981), 34-42.

6. S. Mittal. Design of a Distributed Medical Diagnosis and Database System. Ph.D. Th., Dept. of Comp. and Info. Sci., The Ohio State University, 1980.

7. S. Mittal and B. Chandrasekaran. "Conceptual Representation of Patient Data Bases." J. of Medical Systems (1981).

Proceedings of MEDCOMP '83 (IEEE Computer Society)

A REPRESENTATION FOR THE FUNCTIONING OF DEVICES THAT SUPPORTS COMPILATION OF EXPERT PROBLEM SOLVING STRUCTURES

An Extended Summary

V.S. Moorthy and B. Chandrasekaran
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

(September,1983)

ABSTRACT

Most of the diagnostic systems that have been developed in medicine as well as other domains can properly be called "compiled" knowledge systems in the sense that the knowledge base contains the relationships between symptoms and malfunction hypotheses in some form. However, often in human reasoning, an expert's knowledge of how the device "functions" is used to generate new relationships during the reasoning process. This deeper level representation which can be processed to yield more compiled diagnostic structures is the concern of this paper. Using the example of an household buzzer, we show in this paper what a functional representation of a device looks like. We also indicate the nature of the compilation process that can produce the diagnostic expert from this deeper representation.

1. Introduction

Recently the domain of devices has attracted theoretical as well as applied AI researchers [1, 2, 3, 4, 13, 14,16]. To troubleshoot, modify and monitor devices (eg. nuclear power plant, physiological organs, electronic circuits, computer software, etc.), it is necessary for an agent to represent and use the knowledge about the functioning of the devices. Moreover, an agent needs to know the functioning of similar devices in order to become an expert in designing a new device.

Referring to the "depth" of knowledge in expert systems, Hart [5] and Michie [6] have suggested that systems with deep knowledge will be able to solve problems of significantly greater complexity than the so called surface systems. Their remarks on deep vs. surface systems seem to capture a fairly widespread feeling about the inadequacy of the first generation expert systems. However, in Chandrasekaran and Mittal [7], it is argued that, in principle, given any deep model of a domain, it is possible to compile an expert diagnostic system (more specifically an MDX-like diagnostic system [8, 9, 17]) which is as powerful as the deeper model, but more efficient than the deeper model for diagnostic purposes.

Also, there is no general agreement on the form and content of these deep knowledge structures. Hart [5] suggests that they should model causality by multi-level systems, while Michie, following Rouse [10], proposes that they should represent knowledge of the form "situation x action -> situation". The work of Patil [11] and Pople [12] is based on the idea that the appropriate form for the representation of deep knowledge is a causal net. We propose that with respect to the "surface causality" modeled in systems like MYCIN, the next deeper level to model causality is the functional representation of devices. An agent becomes an expert in various tasks such as diagnosis, design, explanation, etc., by compiling appropriate problem solving structures from the functional representation.

In this paper we describe a representational scheme for the functioning of devices and its utility for compiling an MDX-like [8,9, 17] diagnostic expert system. Our focus here is on the representation; we discuss the compilation in more detail in [15].

2. Comparison with Related Research

De kleer and Brown [1, 2, 3] have been working on the representation of an agent's knowledge about how a device actually functions. This representation, which they call "functional", is actually a causally related sequence of behavioral states, some of which either belong to the components or refer to the attributes of the interconnections between the components. They then proceed to discuss the process of acquiring the above "functional" representation from the structural knowledge of the device. They impose three interesting criteria that such a process, as well as the "functional" representation, should satisfy — namely, "no-function-in-structure", "weak causality" and "strong causality."

Our work differs from that of De kleer and Brown in two aspects: Firstly,our definition of what constitutes a functional representation is different from theirs. Secondly, while they are concentrating on the acquisition of function from structure, we wish to understand the process by which an agent uses the functional representation for various problem solving activities, i.e., transforms the functional representation into "expert" problem solving structures. However, these apparently different objectives are not as disjoint as they might appear. In fact, we strongly believe that our functional representation will ultimately satisfy the twin requirements of acquirability and transformability into expert

problem solving structures.

The functional representation of Davis et al [13] has "functional" as well as "physical" hierarchies. Their "inference rules" and "simulation rules" enable their "violated expectation" approach to troubleshooting. However, their "functional" and "physical" hierarchies neither differentiate nor relate "function","structure","behavior", "assumptions" and "deeper causal Knowledge " as ours.

3. A Representational Scheme for the Functioning of Devices

3.1. The Representational Scheme

Our scheme allows multiplicity of levels in functional representation. The topmost level describes the functioning of a device in terms of the abstractions of its components. The next level describes the functioning of these components using the abstractions of their subcomponents, and so on. As we shall see later, the hierarchy is not just functional. The abstractions from the lower level include, in addition, the states of components as well as other entities.

At each level of our functional representation we propose that there are five significant aspects to an agent's knowledge of functioning of devices:

- STRUCTURE: that specifies the components (subcomponents) of a device (a component) and the interconnections between them.
- FUNCTION: that specifies WHAT is the response of a device or a component to an external or internal stimulus.
- BEHAVIOR: that specifies HOW, given a stimulus, the response is accomplished.
- GENERIC KNOWLEDGE: chunks of deeper causal knowledge that have been compiled from various domains to enable the specification of behavior of devices and their components. For example, a specialized version of Kirchoff's law from the domain of electrical circuits.
- ASSUMPTIONS: under which a behavior is accomplished.

Next we describe the roles of these five aspects in representing the functioning of devices and their notations. Following De kleer and Brown [1, 2], we shall use the household buzzer shown in fig. 3-1 to illustrate our ideas.

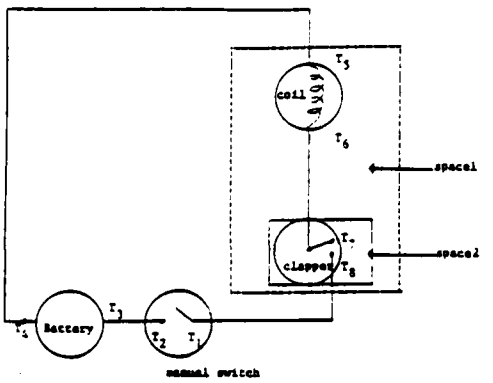


Figure 3-1: A Schematic Diagram of a Household Buzzer

FUNCTION

The functional specification of a device is illustrated below by describing one of the functions of the buzzer.

```
FUNCTION:
 Buzz: TOMAKE buzzing(buzzer)
       IF pressed(manual-switch)*
        PROVIDED assumption2  BY behavior1
```

"buzz" is the name of the function; "buzzing(buzzer)" denotes the buzzing state of the buzzer. "t7" and "t8" are distinguished elements of a component of buzzer (we discuss this below). "assumption2" will specify the initial state i.e., "t7","t8" are electrically connected (more about assumptions later). The "BY" clause relates the function with its behavior i.e., the manner in which the function is accomplished (behavioral specification is described below). As we shall see in section 4.2, this association between function and behavior is important at the compilation stage.

STRUCTURE

The structure of a device (component) is represented using the abstractions of its components (subcomponents) and generic relations between them (such as "serially-connected"). As an illustration consider the structure of the buzzer given below:

```
STRUCTURE:
   COMPONENTS:
        manual-switch (t1,t2), battery (t3,t4),
        coil (t5,t6,space1), clapper (t7,t8,space2)
   RELATIONS:  serially-connected (manual-switch,
                          battery,coil,clapper)
               AND includes(space1,space2)

   ABSTRACTIONS-OF-COMPONENTS:
       COMPONENT  clapper (T1,T2,SPACE)
           FUNCTIONS: mechanical,acoustic,magnetic
           STATES: elect-connected (T1,T2),
                   repeated-hit(clapper)
       END COMPONENT

       COMPONENT  coil (T1,T2,SPACE)
             .
             .
             .
       END COMPONENT
   END ABSTRACTIONS-OF-COMPONENTS
END STRUCTURE
```

"t1","t2"...., "space" are distinguished elements (terminals) of components ; only between distinguished elements can relations be defined. "mechanical","acoustic", etc., are the names of functions of clapper. These functions (as well as the structure, behavior, generic knowledge and assumptions relating to the clapper as well as other of the components) are represented at the next level of our representation in the same manner as the buzzer. The capitalized parameters such as T1,T2,etc., are local to the associated component.

It is important to note the following:

a) A component (subcomponent) is specified independent of the representation of the device (component) which contains it. More specifically, the specification of a component does not refer to the role of the component in the composite. Thus our representation obeys the "no-function-in-structure" principle of De kleer and Brown [1, 2].

b) Not the behavioral specifications of components but only the names of the functions are carried over to the higher level. This property is important when an agent needs to replace a malfunctioning component by a functionally equivalent but behaviorally different one. Note that neither the "intrinsic mechanism" nor the "causal model" of De kleer and Brown [1] distinguishes between function and behavior as we do. The "behavioral description" of Davis et al [13] and Davis [14] is similar to our functional specification. They do not have any construct equivalent to our behavioral specification. (The significance of having a behavioral specification will become clear when we discuss it below.)

c) We model interconnections between components by relations such as 'serially-connected', 'includes', etc.

BEHAVIOR

The behavioral specification of a device describes the manner in which a function is accomplished by "gluing" together the functions of components, generic knowledge, *assumptions* relating to behavioral alternatives, and sub-behaviors. For example, the specification of of 'behavior1' in fig. 3-2 illustrates how the 'buzz' function discussed above is realized.

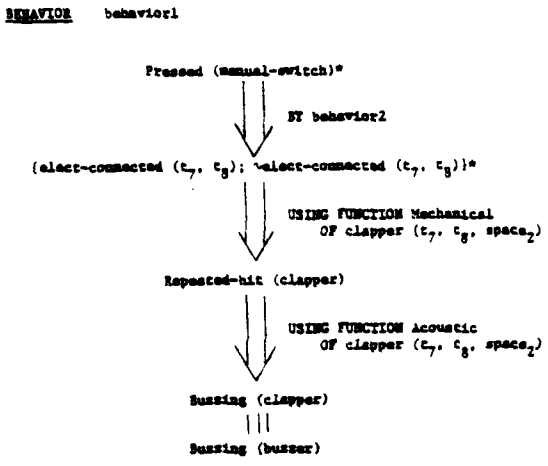


Figure 3-2: An Illustration of Behavioral Specification

We have made use of five conceptually important notations in behavioral specification. They are described below:

```
1:          s1
            ||
            ||  BY <name-of-a-behavior>
            \/
            s2
```

For example,

```
      Pressed (manual-switch)*
              ||
              ||  BY behavior2
              \/
    { elect-connected (t7,t8);  }
    { ~ elect-connected (t7,t8) } *
```

This means that the state s1 causes the state s2 and the details are in another behavioral specification ("behavior2"). This relation enables the the specification of behavior of a component (or a device) at many levels of detail but still at the level of the component (or device).

```
2:    s1
      ||
      ||  USING FUNCTION  <name-of-a-function>
      ||          OF <component>
      \/
      s2
```

For example,

```
repeated-hit (clapper)
      ||  USING FUNCTION acoustic
      ||  OF clapper(t7,t8,space2)
      \/
buzzing (buzzer)
```

The above notation means that the state s2 is caused from s1 by making use of a function ("acoustic") of the component (clapper). This relation makes it possible to glue the functions of the components together to obtain a behavior.

```
3:
      S1 ≡ S2
```

The above notation means that the agent will "equivalence" the state s1 of a component (or subcomponent) with s2, the state of a device (or a component). For example, as in the specification of "behavior1" (ref. to fig. 3-1 "buzzing(clapper)" is "equivalenced" with "buzzing(buzzer)". Note that without this relation, it is impossible to connect the result of a function of a component (the buzzing of the clapper) with the result of a function of the device (the buzzing of the buzzer which is the result of its function "buzz"). Without this connection, "behavior1" cannot be claimed to implement the function "buzz" of the buzzer.

```
4:      s1
        ||
        ||  AS-PER <name-of-a-knowledge-chunk>
        ||    IN-THE-CONTEXT-OF  < one or more
        \/       of a "relation", "state" or a
        s2       "function of a component" >
```

For example,

```
elect-connected(t7,t8)
      ||
      ||  AS-PER knowledge1 IN-THE-CONTEXT-OF
      ||   FUNCTION  voltage OF battery(t1,t2),
      ||    serially-connected(battery,coil,
      \/    clapper,manual-switch)
voltage-applied(t5,t6)
```

This means that if the terminals t7 and t8 are electrically connected, then voltage will be applied between t5 and t6. This is true as per the knowledge chunk called 'knowledge1' when it is applied in the context of battery, coil, clapper and manual switch being serially connected, and the battery makes voltage available at its terminal. (The representation of 'knowledge1' is discussed *below.) It is through this primitive that the role* of generic knowledge in describing a behavior is represented.

```
5:    magnetized(space2)
         ||
         ||  USING FUNCTION magnetic
         ||     OF clapper(t7,t8,space2)
         ||     WITH assumption3
         \/

      ~elect-connected(t7,t8)
```

"assumption3" will specify that there exists a force F such that if space2 is magnetized, then the resulting magnetic force will be greater than F. Note that "assumption3" does not specify what is F, how it is to be realized and so on. The "WITH" clause, like "PROVIDED", relates an assumption with the state transition. However, "WITH" is different from "PROVIDED" since it relates an assumption that is passed from a device (component) to a component (sub-component) while "PROVIDED" relates the one *from a component* (sub-component) to the device (component). Also, assumptions related by "WITH" clause can be used to make a state transition deterministic.

GENERIC KNOWLEDGE

The generic knowledge specification of a device (component) describes all chunks of deeper knowledge used in its behavioral specification.

The following is a specification of 'knowledge1'.

```
GENERIC KNOWLEDGE:
   knowledge1:

      voltage-applied (t1,t2)
           ||
           || AS-PER kirchoff's-law
           ||  IN-THE-CONTEXT-OF
           ||  elect-connected(t1,t3)
           ||   ^ elect-connected(t2,t4)
           \/
      voltage-applied (t3,t4)
```

It is worth noting that the specification of generic knowledge is context-free. The context in which it is applied is specified in the behavioral specification (as illustrated above). As we shall see soon, there is a mechanism ("REFERENCES") by which a user task of the functional representation knows where to look for the definition of Kirchoff's law.

We would like to draw particular attention to the notion of GENERIC KNOWLEDGE in our representation. This enables us to capture the relation between functional representation and deeper causal knowledge. Moreover, without an

explicit link with such generic knowledge it is not possible to support the recognition of incorrect application of such knowledge during "envisioning" [1,2,3]. Also it cannot support queries relating to the role of such knowledge in understanding,describing,explaining,etc., of the behavior of devices.

ASSUMPTIONS

All assumptions made use of in the behavioral specification of a device (component) are described in ASSUMPTIONS as illustrated below with reference to the clapper.

```
ASSUMPTIONS:
  DEFINITIONS:

    f1 = magnetic-force DUE-TO magnetized(space) AND
    f2 = spring-force DUE-TO loaded (spring)

  ASSUMPTION1:
    IF magnetized(space)  THEN  f1 > f2

  ASSUMPTION2:
    IF  ~ magnetized(space)  THEN  f1 < f2

END ASSUMPTIONS
```

"spring-force" and "magnetic-force" are concepts; we discuss below about their definition. The primitive "DUE-TO" relates a concept with a state of a component.

Note that though De kleer and Brown [2] state that a difference between a novice and an expert is that the latter has made explicit all the assumptions underlying behavior of devices, their causal model, unlike our functional representation , does not represent explicitly the role of assumptions in behavior.

REFERENCES

Clearly an agent's knowledge of the functioning of devices will have references to elements of different domains ,e.g., electrical circuits,electro-magnetism ,etc. These references are specified in the "REFERENCES" part of our representational scheme as illustrated below:

```
REFERENCES:

   FOR kirchoff's-law, elect-connected
        REFER-TO elect-circuits
   FOR magnetic-force  REFER-TO electro-magnetism

END REFERENCES
```

Note that we do not yet know how to represent domains such as "elect-circuits", "electro-magnetism",etc.

4. Compilation of a Diagnostic Expert

The principal function of the compiler that we shall discuss here is to generate a diagnostic expert from the functional representation. Checking the correctness/ consistency of a functional representation, optimization of the generated expert systems are also significant aspects of the compilation process. However, for want of space , we discuss here only the generation of an MDX-like diagnostic expert. Other aspects of compilation are discussed in [15].

4.1. The Structure of the Generated Diagnostic Expert System

As shown in fig. 4-1, the generated expert is a hierarchy of specialists. Each specialist corresponds to a malfunction in the device at a certain level of abstraction. For example, a bad clapper, bad serial connection, etc. Specialists corresponding to more general or abstract malfunctioning are higher in the hierarchy. For example,the root specialist in fig.4-1 corresponds to a "malfunctioning buzzer". Its three sub-specialists correspond to the following three malfunctions (only the first one is shown in fig. 4-1):

1. The buzzer does not buzz when the manual switch is pressed.
2. A buzzing buzzer does not stop buzzing when the manual switch is released.
3. The buzzer keeps buzzing independent of the state of the manual switch.

Every specialist has knowledge to establish the associated malfunctioning and to refine it by calling its sub-specialists. The knowledge of a specialist is in the form of three types of rules: confirmatory rules,exclusionary rules and recommendations. (We will not discuss "recommendations" here since it is concerned with optimization of the generated expert.) For example,

```
IF elect-connected (t1,t2) ^
  ~ voltage-applied (t5,t6)
    THEN  confirm
IF voltage-applied (t5,t6) THEN reject
```

A malfunction is diagnosed top-down by establishing a specialist and refining the malfunction represented by it by calling its sub-specialists. This discussion of the structuring and functioning of the diagnostic expert is grossly simplified. More detailed information can be obtained from [8, 9].

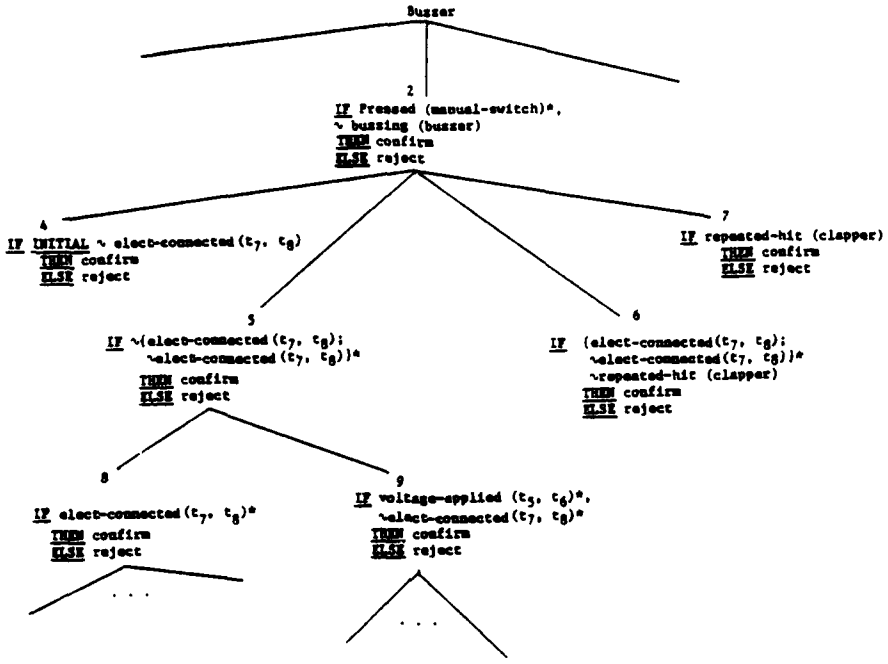


Figure 4-1: An Example of a Generated Diagnostic Expert

There are three types of malfunctioning and hence three types of specialists:

1. An assumption might have been violated. The specialist associated with it is called an assumption checker.
2. A function may not be functioning correctly. The associated specialist is called a function checker.
3. A relation between components may not hold. For example, the battery, coil and clapper may not be connected serially. Let us call the specialist the relation checker.

4.2. The Compilation Process

At the start, the compiler generates the root specialist. The root specialist needs no knowledge to establish itself. The fact that the expert is invoked leads automatically to the establishment of the root specialist. The compiler then processes the various functions of the device and generates a function checker corresponding to each function. For example, given

```
FUNCTION:
  buzz: TOMAKE  buzzing(buzzer)
          IF pressed(manual-switch)* BY behavior1
```

the compiler will generate a function checker with the following rule:

```
IF  pressed(manual-switch)* ^ ~buzzing(buzzer)
    THEN confirm
    ELSE  reject
```

Then the function checkers generated as above will be attached to the root specialist. Afterwards the compiler, using the "BY" clause, obtains the behavior associated with each function and compiles it. For example, if the behavior is specified in the form:

s1 ⟹ s2 ⟹ ..... ⟹ sn

then the compiler will generate a set of n-1 specialists for the function checkers associated with the behavior. The rules for them will be:

```
IF s1 ~~ s2     THEN confirm
                ELSE reject
                .
                .
                .
IF sn-1 ~~ sn   THEN confirm
                ELSE reject
```

For the "buzz" function example given above, nodes 5, 6 and 7 in fig. 4-1 will be generated using "behavior1" in fig.3-2. Note that the rule associated with node 5 should be:

```
IF pressed(manual-switch)*
 ^(elect-connected(t7,t8);~elect-connected(t7,t8))*
    THEN confirm
    ELSE reject.
```

However, the condition "pressed(manual-switch)*" is not checked since it is done at node 2, i.e., the parent of the node 5.

Further processing of a behavioral specification depends on the kind of composition of behavior.

CASE 1:

Assume that node 5 is generated corresponding to the following state transition in fig. 3-2.

```
    Pressed(manual-switch)*
        ||
        ||        BY  behavior2
        \/
(elect-connected(t7,t8) ; ~elect-connected(t7,t8))*
```

The above state transition will also result in compiling 'Behavior2' as described above, and attaching the generated specialists to node 5.

CASE: 2

Let the state transition in fig.3-2corresponding to node 6 in fig.4-1 be:

```
(elect-connected(t7,c8); ~elect-connected(t7,t8))*
      ||
      ||   USING FUNCTION mechanical
      \/       OF clapper(t7,t8,space2)
  repeated-hit(clapper)
```

After generating the node 6 for the above state transition, the compiler will look at the function 'mechanical' in the functional representation of the component 'clapper' and compile the behavior associated with the function. If there is no behavioral specification for the function, node 6 will be a tip specialist. If there is one for the function, then it will be compiled, and the generated specialists will be attached to node 6. If the function of the component is implemented under, say, "assumption1" and the specification of "assumption1" is of the form:

```
IF s3 THEN s4
```

then the additional specialist with the rule

```
IF s3 ^ ~s4 THEN confirm
            ELSE reject
```

will be generated and attached to the node corresponding to the above transition. An example of an assumption checker is node 4 in fig.4-1.

CASE 3:

The state transition

```
        s1
        ||  AS-PER knowledge1
        ||   IN-THE-CONTEXT-OF s3^s4...sn
        \/
        s2
```

will result in a set of sub-specialists with the rules:

```
IF ~s3 THEN confirm
       ELSE reject

IF ~s4 THEN confirm
       ELSE reject
   .
   .
   .
IF ~sn THEN confirm
       ELSE reject
```

REFERENCES

1: De Kleer, J., Brown, J.S., "Mental Models of Physical Mechanisms and Their Acquisition," in Cognitive Skills and Their Acquisition, (ed) J.R. Anderson., Erlbaum, 1981.

2: De Kleer, J., Brown, J.S., "Assumptions and Ambiguities in Mechanistic Mental Models," tech rept. CIS 9, Xerox Palo Alto Research Centers, 1982.

3: De Kleer, J., Brown, J.S., "Foundations of envisioning," Proceed. of AAAI, 1982.

4: Kuipers, B., De Kleer and Brown 'Mental Models', A Critique," TUWPICS No. 17, Working Papers in Cognitive Science, Tufts Univ., 1981.

5: Hart, P.E., "Directions for I in the Eighties," SIGART newsletter, no. 79, Jan. 1982.

6: Michie D., "High-road and Low-road Programs," AI magazine, 3:1, 1982.

7: Chandrasekaran, B., and Mittal S., "Deep vs. Compiled Knowledge Approaches to Diagnostic Problem Solving," Proc. Second National AI Conf., AAAI, Pittsburgh, PA., 1982., revised version to appear in International Journal of Man Machine Studies, 1983.

8: Gomez, F. and Chandrasekaran, B., "Knowledge Organization and Distribution

for Medical Diagnosis," IEEE Trans. Systems, Man and Cybernetics, SMC-11:1, 1981.

9: Chandrasekaran, B., "Decomposition of Domain Knowledge into Knowledge Sources: The MDX Approach," Proc. of Fourth National Conf. of Canadian Society for the Computational Studies Of Intelligence (CSCSI/SCEIO), 1982., Revised in "Towards a Taxonomy of Problem Solving Types," AI magazine, Winter/Spring , 1983.

10: Rouse, W.B., and Hunt, R.M., "A Fuzzy Rule-Based Model Of Human Problem Solving in Fault Diagnosis Task," Working Paper, Coord. Science Lab, Univ. Illinois, Urbana, 1980.

11: Patil, R.S., "Causal Representation of Patient Illness for Electrolyte and Acid Base Diagnosis," Ph.D. Dissert., TR-267, MIT Lab for Computer Science, Cambridge, Mass., 1981.

12: Pople H.E., "Heuristic Methods For Imposing Structures On Ill-Structured Problems," Artificial Intelligence In Medicine, ed. by Szolovits, Westview, 1982.

13: Davis R., Shrobe H., Hamscher W., Wieckert, K., Shirly M., Polit S., "Diagnosis Based on Description of Structure and Function ," Natnl conf. on AI, 1982.

14: Davis R., "Diagnosis via Causal Reasoning: Paths of Interaction and the Locality Principle," Natnl. conf. on AI , 1983.

15: Moorthy V.S., and Chandrasekaran B., "A Functional Representation of Devices and Compilation of Expert Problem Solving Systems", Tech. report, AI group, Dept of Comp and Info Science, Ohio State University, Sept.,1983.

16: Chandrasekaran B., "Expert System: Matching Techniques to Tasks", Invited Presentation at the Newyork Univ. Symposium on "Artificial Intelligence Applications for Business", May 1983.

17: Chandrasekaran B., Mittal S., "Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and related Systems", in Advances in Computers Vol.,22, Academic Press,Inc. 1983.

ACKNOWLEDGEMENTS

Research was supported in part by AFOSR Grant 82-0255 and NSF Grant MCS 8103480.

APPENDIX

To the paper

"A REPRESENTATION FOR THE FUNCTIONING OF DEVICES
THAT SUPPORTS COMPILATION OF EXPERT PROBLEM SOLVING STRUCTURES"

by V.S. Moorthy and B. Chandrasekaran

Details of the functional representation of FUNCTION: buzz of the buzzer

**NOTE: We have represented below only the buzzer;
Battery,coil,clapper and manual switch have NOT been represented.**

```
DEVICE   buzzer
   FUNCTION:
        buzz: TOMAKE buzzing (buzzer)
              IF pressed (manual-switch)*
              PROVIDED  INITIAL elect-connected (t7,t8)
              BY behavior1

        stop-buzz:TOMAKE  ~buzzing (buzzer)
                  IF  ~pressed (manual-switch)
                  PROVIDED INITIAL buzzing (buzzer)
                  BY behavior5


   STRUCTURE:
        COMPONENTS:
            manual-switch (t1,t2), battery (t3,t4),
            coil (t5,t6,space1), clapper (t7,t8,space2)

        RELATIONS:
            serially-connected (manual-switch,battery,coil,clapper),
            includes (space1,space2)

        ABSTRACTIONS-OF-COMPONENTS:
            COMPONENT clapper (T1,T2,SPACE)
                FUNCTIONS: mechanical,acoustic,magnetic
                STATES: elect-connected (T1,T2),
                        repeated-hit (clapper)
            COMPONENT coil (T1,T2,SPACE)
                FUNCTIONS: magnetic
                STATES: magnetized (SPACE), voltage-applied(T1,T2)

            COMPONENT  manual-switch(T1,T2)
                FUNCTIONS: connect
                STATES: elect-connected (T1,T2),
                        pressed (manual-switch)

            COMPONENT  battery (T1,T2)
                FUNCTIONS: voltage
```

BEHAVIOR:

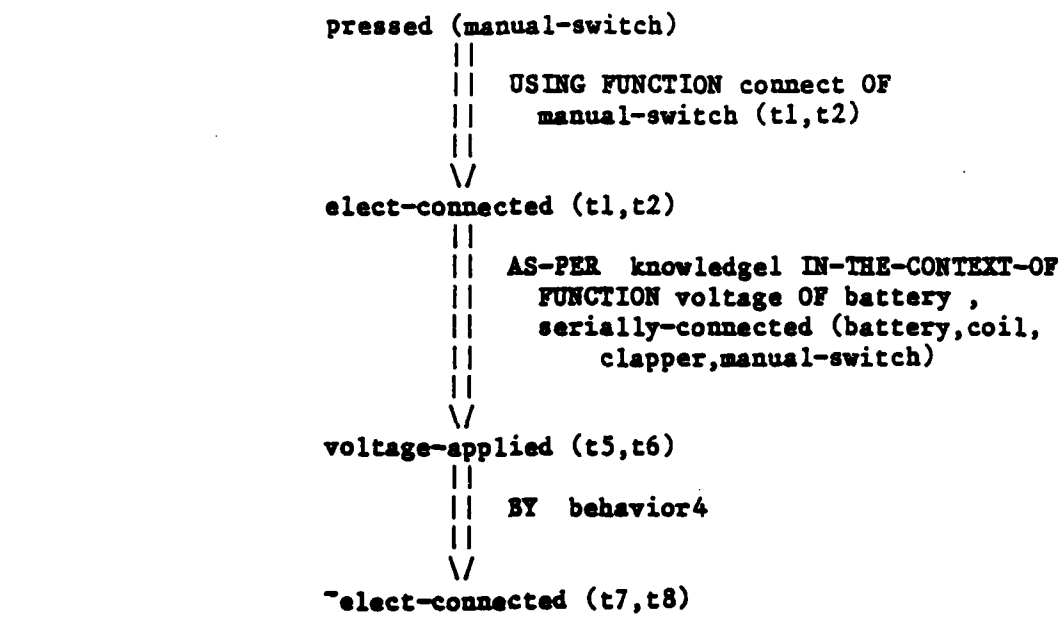
behavior1:

```
        pressed (manual-switch)*
                ||
                ||  BY behavior2
                ||
                \/
{ elect-connected (t7,t8);  elect-connected (t7,t8)} *
                ||
                ||  USING FUNCTION mechanical OF
                ||      clapper(t7,t8,space1)
                ||
                \/
        repeated-hit (clapper)
                ||
                ||  USING FUNCTION acoustic OF
                ||      clapper (t7,t8,space2)
                ||
                \/
        buzzing ( clapper)
                |||
                |||
        buzzing (buzzer)
```

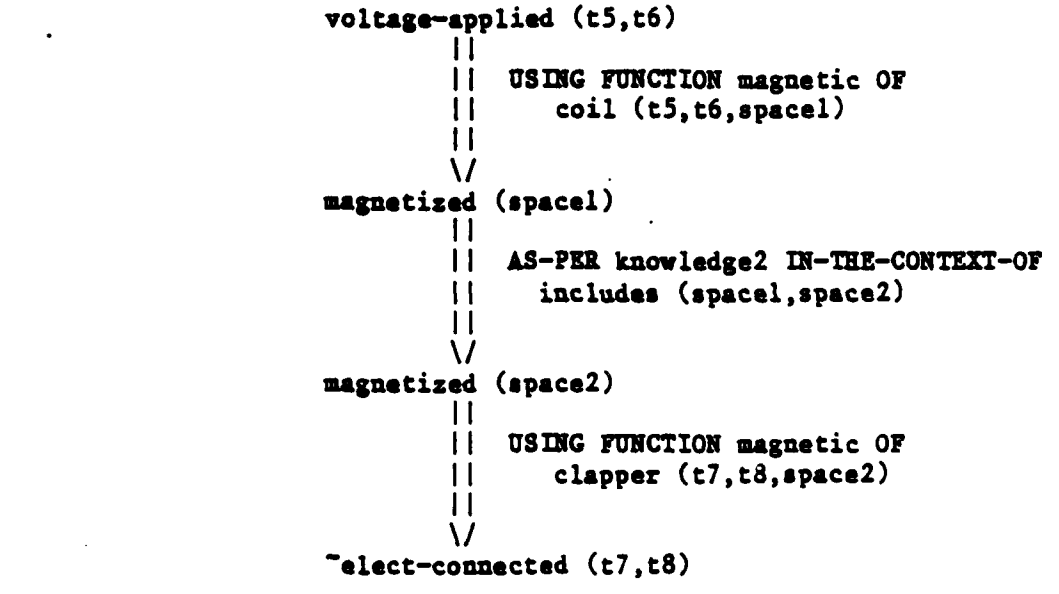
behavior2:

```
   {    pressed (manual-switch)
                ||
                ||  BY behavior3
                ||
                \/
        ~elect-connected (t7,t8)
                ||
                ||  AS-PER knowledge1 IN-THE-CONTEXT-OF
                ||      serially-connected (battery,coil,
                ||              clapper,manual-switch) ^
                ||      FUNCTION voltage OF battery
                \/
        ~voltage-applied (t5,t6)
                ||
                ||  BY behavior4
                ||
                \/
        elect-connected (t7,t8) } *
```

behavior3:



behavior4: IFF



```
GENERIC KNOWLEDGE:
    knowledge1:

                Voltage-applied (t1,t2)
                        ||
                        ||  AS-PER kirchoff's-law
                        ||    IN-THE-CONTEXT-OF
                        ||    elect-connected (t1,t3),
                        ||    elect-connected (t2,t4)
                        ||
                        \/
                voltage-applied (t3,t4)


    knowledge2:

                magnetized (space1)
                        ||
                        ||  AS-PER laws-of-space
                        ||   IN-THE-CONTEXT-OF
                        ||      includes (space1,space2)
                        ||
                        \/
                magnetized (space2)




ASSUMPTIONS:
   DEFINITIONS:
        f1= magnetic-force DUE-TO magnetized (space)
        f2= spring-force  DUE-TO  loaded (spring)

    assumption1:
        IF magnetized (space) THEN f1 > f2

    assumption2:
        IF ~magnetized (space) THEN f1 < f2


END-DEVICE buzzer
```

To appear in a book on Expert Systems ~~to be~~ edited by W. Reitman, Ablex Corp., publishers.

EXPERT SYSTEMS: MATCHING TECHNIQUES TO TASKS

B Chandrasekaran[1]
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

ABSTRACT

In this paper an attempt is made to relate the architectures and representations for expert systems to the types of tasks for which they are appropriate. We start with an analysis of the features that characterize an expert system, and discuss the need for symbolic knowledge structures that support qualitative reasoning in the design of expert systems. We consider rules, logical formulas and frames for representation of expert knowledge. In particular we provide an analysis of the multiplicity of roles that rules have played in different rule-based systems, and emphasize the need to distinguish between these rules. We proceed to outline our theory of types of expert problem solving and argue that such a taxonomy enables one to characterize expert system capabilities and help match problems with techniques. Throughout the paper it is emphasized that the important issue is the nature of the information processing task in a given task domain, and issues of formalisms for representation are subordinate to that basic issue.

---

[1]Research supported by AFOSR Grant 82-0255

TABLE OF CONTENTS

1. EXPERT SYSTEMS: WHAT ARE THEY?

It is clear that recently artificial intelligence has created enormous excitement in the commercial marketplace, and one of the sources of this excitement is the promise of expert systems or knowledge based systems in solving or assisting in the solution of many practical problems. There is a widespread feeling that knowledge is the next frontier in the practical application of computers, and the well-publicized commitment of the Japanese to research on and development of a Fifth Generation computer for knowledge processing has only added to this sense of an impending revolution. The phrase "expert systems" evokes all sorts of hopes: From the clerical worker to the research scientist in a corporation, each employee is a storehouse of an enormous amount of knowledge and problem solving capabilities. The syllogism goes something like, "I need an expert for X, they are hard to find or expensive, thus I need an expert system for X." It is clear to most of us doing research in the field that while the promise of carefully deployed expert systems is indeed high, we are nowhere near the stage where we can hope to replace all kinds of human experts with computer-based expert systems. To take an extreme example, advanced researchers or creative mathematicians clearly perform knowledge-based expert problem solving, but equally clearly the state of the art in expert systems is not up to replacing them with expert systems. Any attempt to characterize, even if informally, what sorts of problems are amenable to current techniques — matching techniques to tasks, if you will — could be very useful.

A related problem is that, in strict definitional terms, it is hard to be very precise about what characteristics of a computer program to solve a class

of problems qualify it to be called an expert system? The following dimensions have often been suggested as important in such a definition.

- Expertise : Certainly a necessary condition is that an expert system should have expertise in the domain and show expert-level performance is some aspects of the domain. However, this condition is not sufficient. Is a payroll program written in Cobol an expert system? In some real sense it captures the expertise of an accountant whose domain knowledge is incorporated in the many branching decisions made by the logic of the program.
- Search: The intuition that a program must do some search in a space of possibilities — following the idea that search is an essential characteristic of intelligent reasoning — is generally useful but not always valid, because R1 (McDermott, 1982), an expert system that has been very successful in practical use, does not do any search in the execution of its main task.
- Uncertainty: While uncertainty in data or knowledge gives many expert problem domains (such as medicine) interesting additional properties and makes them challenging for the designer of expert systems, it is not a defining characteristic, since, again using R1 as an example, its knowledge and data do not involve probabilistic or other types of uncertainty.
- Symbolic knowledge structures: Most expert systems in the AI field have their domain knowledge in explicitly symbolic form as collections of facts, rules, frames etc., which are explicitly manipulated by problem solving or inference mechanisms to produce answers to questions. This is to be contrasted with mathematical or

simulation models or mainly numerical information as its main knowledge base. However, many well-known expert systems such as Internist represent the bulk of their core knowledge in the form of numerical relations between entities. As an incidental observation, we may note that the general emphasis in expert systems work on symbolic knowledge structures often elicits another sort of response. Engineers trained in mathematical modelling techniques find it difficult to understand why a computer program which evaluates say a system of complex mathematical equations describing some process (such as nuclear reaction in a reactor vessel) is not an expert system — after all, they argue, such a system is an embodiment of very highly specialized expert knowledge, capable of providing answers to a number of questions. Further there is a tendency among some people in this group to regard AI programs as "approximate," because the rules that an expert system will use are supposed to be only heuristic, while the mathematical models are exact. (See sec. 3.2 for a discussion of this issue.)

- Explanation capability: Continuing in the vein of searching for definitional characteristics of expert systems, the idea that such systems should be able to explain their reasoning is a useful constraint on the structure and functioning of expert systems, but as a rule, since the activity of explanation itself is poorly understood, it is not yet certain what structures and functions this requirement rules out or permits.
- Other features: Many people in the industrial world have virtually taken to defining expert systems as those that have a knowledge base

and a separate inference engine, and that knowledge base in expert systems should necessarily be in the form of rules. Again, while rules have been a dominant method for knowledge representation in many first generation expert systems, many expert systems have used frame structures (Pauker, Gorry, Kassirer, Schwartz, 1976), or network structures (Duda, Gaschnig, Hart, 1979). In any case, this emphasis on knowledge representation formalisms often obscures the more fundamental issues of the content of the computation these systems do. For example, Szolovits and Pauker (1978) point out that MYCIN, the system most widely thought of as a prototypical example of the rule-based approach can be recast as a frame-based system which uses procedural attachments to fill the slots in various frames.[2] Taking up the knowledge base/ inference engine separation issue, it is unlikely that a complete separation of knowledge and inference is viable as a basic principle in the organization of expert systems. We (Gomez and Chandrasekaran, 1981; Chandrasekaran, 1982) as well as, more recently, others (Davis, 1982; Stefik, et al, 1982) have argued that knowledge and its use are likely to be more strongly intertwined as the difficulties and variety of the tasks

[2] There is a general problem in AI of not making clean distinctions between the basic information processing task of a computation and the algorithm or program that carries out this task. Marr (1976) presents arguments for such a distinction in computational theories of vision. In Gomez and Chandrasekaran (1981) we make analogous arguments in the area of knowledge representation for problem solving systems. Saying that System A uses rules, while System B uses networks for knowledge representation says nothing about the nature of the information processing activities that go on in the two systems: a comparison at the formalism level would miss many important aspects of the similarities and differences that ought to be sought at a higher level. Our discussions later in the paper regarding rules elaborate on some aspects of this point.

the expert systems are called upon to perform increase. (This argument will be elaborated in Sec. 7 of this paper.)

The point of the foregoing is not to suggest a precise, issue-settling definition of expert systems, but merely to point to the multiple dimensions along which expert systems can be viewed, and to the need for more careful analysis of much of the terminology that is used in discussing expert systems.

The major line of argument that we will pursue in this paper can be outlined as follows. In Sec. 2, we briefly trace the development of the idea of knowledge-based systems in AI. Sec. 3 is devoted to discussing the increasing need for symbolic content to expert reasoning as the size and demands of the task domain increase; i.e, we will analyze why a complete mathematical model of the situation, even if available, will not meet many of the demands placed on expert reasoning. In Sec. 4, we discuss the several distinct senses and roles that the notion of rules can play and have played in expert systems, and how a failure to keep these separate can cause a great deal of confusion. In Sec. 5, we briefly discuss logic-based and frame-based representations. In Sec. 6, we discuss the need for organization of knowledge for effective use, and in Sec. 7, we argue that further organizational constructs, such as concepts and types of problem solving, are needed both to construct more powerful expert systems, and to characterize their capabilities. This will also have the side effect of emphasizing the increasing need not to separate knowledge bases and inference machineries. We will also provide in this section two examples of generic problem-solving types, and show how each type of problem-solving induces an organization of knowledge in the form of a cooperating community of "specialists" engaged in

that problem solving type. At this point, we will be able to discuss what sorts of problems can be handled by "compiled" structures and what sorts need "deeper" problem solving. In that context we will discuss the issues related to the so-called causal modeling problem. This will help us present a discussion of the the issues surrounding the degree of understanding that expert systems may need to have about the domain. The overall flow of the discussion is in the direction of the evolution of expert systems from numerical programs to highly organized symbolic structures engaged in distinct types of problem-solving and communicating with one another.

An admission is in order at this point. The title is more ambitious than what we will be able to accomplish in this paper. A really satisfactory account must await a better overall theory of problem solving than we have at present. Even an incomplete theory such as the one presented in Sec. 7 is capable of providing a framework for characterizing capabilities of expert systems in generic terms. Thus the paper should be viewed as stating a position on the sorts of theories that might help us characterize in powerful ways how the "engineering" of knowledge might rest on a more systematic understanding.

2. ON THE IDEA OF KNOWLEDGE-BASED SYSTEMS

A historic reminder may be useful to clarify the term, "knowledge-based." This phrase, in the context of AI systems, arose in response to a recognition, mainly in the pioneering work of Feigenbaum and his associates in the early to mid-70's, that much of the power of experts in problem solving in their domains arose from a large number of rule-like pieces of domain knowledge which helped constrain the problem solving effort and direct it towards potentially useful intermediate hypotheses. These pieces of knowledge were domain-specific in the sense that they were not couched in terms of general principles or heuristics which could be instantiated within each domain, but rather were directly in the form of appropriate actions to try in various situations. The situations corresponded to partial descriptions stated in terms of domain features . This hypothesis about the source of expert problem solving power was in contrast to the previous emphasis in problem solving research on powerful general principles of reasoning which would work on different domain representations to produce solutions for each domain. Thus the means-ends heuristic of the General Problem Solver program (Newell and Simon, 1972) is a general purpose heuristic, which would attempt to produce solutions for different problems by working on the respective problem representations. On the other hand, a piece of knowledge like, "When considering liver diseases, if the patient has been exposed to certain chemicals, consider hepatitis," is a domain-specific heuristic which the human expert was said to use directly.

The paradigm for knowledge-based systems that was elaborated consisted of extracting from the human experts a large number of such rules for each domain

and creating a knowledge-base with such rules. It was generally assumed as part of the paradigm that the rule-using (i.e., reasoning) machinery, was not the source of problem solving power, but rather the rules in the knowledge base. Hence the slogan, "In Knowledge Lies the Power."

The word "knowledge" in "knowledge base systems" is used in a rather special sense. It is meant to refer to the knowledge an expert problem solver in a domain was posited to have which gives a great deal of efficiency to the problem solving effort. The alternative against which this position is staked is one in which complex problem solving machineries are posited to operate on a combination of basic knowledge that defines the domain with various forms of general and common sense knowledge. Thus the underlying premise behind much of the AI work on expert systems is that once the body of expertise is built up, expert reasoning can proceed without any need to invoke the general world and common sense knowledge structures. If such a decomposition were not in principle possible, then the development of expert systems will have to await the solution of the more general problem of common sense reasoning and general world knowledge structures. What portion of expert problem solving in a given domain can be captured in this manner is an empirical question, but experience indicates that there is a nontrivial subset of expert problem solving in important domains that can be captured in this manner. This decoupling of common sense and general purpose reasoning from domain expertise is also the explanation for the rather paradoxical situation in AI where we have programs which display, e.g., expert-like medical diagnostic capabilities while the field is still some distance from capturing most of the intellectual activities that children do with ease.

We will argue later in the paper that while many first generation expert

systems have been successful in relatively simple problems with this approach which lays relatively greater emphasis on heuristic knowledge specific to the domain at the expense of the problem solving aspects, the next generation of research in this area — as well as application systems — will be bringing back an increased emphasis on the latter.

3. FROM NUMERICAL TO SYMBOLIC MODELING OF EXPERTISE

In this section we will provide two sets of reasons why symbolic structures become necessary to support decision-making. By considering the example of multivariate prediction, we will suggest that certain kinds of computational problems are alleviated by multiple-level symbolic structures. In the next subsection, we will argue that certain kinds of decisions cannot be made purely within a numerical model however complete it may be in principle.

3.1. Multivariate Classification

There is an ubiquitous but conceptually simple class of problems in decision-making which can often be typically modeled as mapping a multivariate state vector to a set of discrete categorical states. The classical pattern classification paradigm deals with this class of problems. There are many application domains in which such problems arise locally, and domain experts may be called upon to perform such a classificatory task as part of a larger problem-solving effort. Classifying weather conditions based on a number of measurements and predicting the presence or absence of certain pathophysiological conditions on the basis of a vector of numerical predictor variables are examples of this task. Often, once the predictor variables

themselves are chosen in consultation with the experts, and when the state vector is of modest dimensionality (in the order of 10's), a discriminant type of analysis can be as good as or better than human experts. Much theoretical effort in pattern recognition notwithstanding, experience in this type of task was that this is an example where power came from expert choice of the variables, rather than in the complexity of the discriminants themselves. But when the dimensionality of the state vector gets very large this approach has serious problems both in one's ability to compute the discriminant as well as in the sensitivity of the decision to small changes. Using the medical example earlier, the totality of the medical diagnostic problem can be formally viewed as a mapping from a (very large) vector of manifestations to a (again quite large) set of named diseases. What works very well locally, i.e., for state vectors of low dimensionality and few decision states, deteriorates rapidly when the sizes get large. It doesn't matter which sort of discriminant one uses: statistical ones or perceptron-like threshold devices. The computational problems in the statistical case are described well in Szolovits and Pauker (1978). Corresponding difficulties, especially those relating to sensitivity issues, for perceptron-like devices are discussed by Minsky and Papert (1969).

One approach to overcoming the above computational problems is to introduce multiple <u>layers</u> of decisions. Instead of one classification function from say a 200-dimensional state vector, the problem can be broken into groups of (possibly overlapping) state vectors of the order of 10's, each of which providing a small number of discrete values as local mappings. Typically these groupings will correspond to potentially meaningful

intermediate entities, so that one can interpret each grouping as computing a symbolic abstraction corresponding to an intermediate concept. At the second layer, the outputs of each of these can be grouped together in a similar manner and the process repeated. This is precisely what Signature Tables of Samuel (1967), did in transforming a description of a checker board configuration into a classification in terms how good the board was for a player. Each stage of the abstraction is computationally simple. An important point to notice is that a shift away from numerical precision towards discretization and symbolization is taking place here in capturing expertise.

3.2. Formal Models vs Symbolic Knowledge Structures

Even if one had a complete mathematical model of a situation, that by itself is not often sufficient for many important tasks. All the numerical values of the various state variables will still need to be interpreted. Identifying interesting and potentially significant states from the initial conditions requires qualitative reasoning rather than a complete mathematical simulation. To take a pedagogically effective but fanciful example, consider a household robot watching its master carelessly move his arm towards the edge of a table where sits a glass full of wine. In theory a complete mathematical equation of the arm and all the objects in the room including the volume of wine is possible. But the most detailed numerical solution of this will still only give values for a number of state variables. It still takes further reasoning to interpret this series of values and arrive at a simple common sense level statement, viz., "the arm will hit the wine glass and wine will

spill on the carpet." On the other hand the aim of "naive physics" models is to support qualitative reasoning that can arrive at such conclusions readily. The reason why a complete numerical solution is not enough, i.e., why the above symbolic conclusion ought to be reached by the robot is that its own knowledge of available actions is more appropriately indexed by symbolic abstractions as "spilled wine," rather than by the numerical values of all ranges of relevant state variables in the environment.

Thus when faced with reasoning tasks involving complex systems, both human experts as well as expert systems are necessarily compelled to deal with symbolic knowledge structures, whether or not complete mathematical models may in principle be available. Human experts (as well as AI expert systems) may, at specific points during their qualitative reasoning, switch to a local formal analysis, such as a medical specialist using formulas or equations to decide which side of the acid/base balance a patient may be in, but this formal analysis is at the overall control of a symbolic reasoning system. It is the symbolic knowledge and problem solving structures that are of central interest to the science and technology of AI.

4. ON THE ROLE OF RULES IN RULE-BASED SYSTEMS

As anyone with even a cursory knowledge of expert systems literature would know, the most dominant form of symbolic knowledge in the first generation of expert systems has been rules. In our view, the idea of rules as a formalism for encoding knowledge comes from at least three distinct traditions, and a failure to distinguish between the different senses implied by them is often a great source of misunderstanding. Let us list the three senses.

4.1. Rule Systems as "Universal" Computing Systems

It is well-known that Post productions as well as Markov Algorithms, both of which are rule-based formalisms are examples of universal computation systems. Any computer program, including an expert system, can be encoded in one of these rule systems with only some minimal constraints on the interpreter. In this sense of the term, the rule-based approach becomes a programming technology. Some of the rules in almost every major rule-based system perform such a purely programming role. For example, rules in R1 (McDermott, 1982) which set contexts can, in another representation, be simply viewed as a call for a module which contains a block of knowledge relevant to that context. Metarules (Davis, 1976) can also be viewed as attempts to enable the specification of control behavior in a rule-based programming system. Not all algorithms can be equally naturally encoded in rule formalisms, however. Thus often expert system designers who build rule-based systems complain of frustrations they face when they have to come up with rules to make the system have good control behavior, or to express all domain knowledge in rule form; i.e., they are not encoding "domain knowledge" as much as doing programming in a rule-system.

Whether the rule-based approach is a good programming technology for an expert task depends on both whether the necessary control behavior as well as the "facts of the domain" can be most naturally represented in rule form. In some domains there is a 20%/80% effect — i.e., a large percentage of the domain knowledge may appear to be capturable by a relatively small number of rules, but a rapid growth in the number of rules required sets in if one attempts to capture more and more of the domain knowledge (McDermott, 1982).

4.2. Rules as in "Rules-of-Thumb"

The common sense notion of rules is one of an approximate, quick guide to action, a computationally less expensive alternative to real thinking, adequate for most occasions, but still with potential pitfalls. A related notion is that of a rule as something that captures a relationship which is only statistically valid, or a relationship whose causal antecedents are poorly understood. Rules in medical diagnosis systems of the form, "If male and findings x, y, z, assign k units of evidence to hypothesis H," are of this type, where some statistical differential between the sexes in the likelihood of having a certain disease might be used.

This sense of rules is often the basis of concerns that rule-based approaches are "shallow" and for hard problems "causal" models are needed.

It ought to be emphasized that, this common sense notion notwithstanding, the fact that an expert system is rule-based should not necessarily imply that it is engaged in shallow reasoning, or that it is using knowledge of only approximate validity. For example, R1 uses rules which are perfectly sound pieces of knowledge about the domain of computer system configuration. To the extent that any computer program can be written in a rule-based computational framework, the shallow vs. deep characterization does not arise from its rule form, but from the character of the rules.

4.3. Rules as Cognitive Units

The third stream of ideas relating to the prominence of rules in expert systems is the idea, due to Newell (1983), of rules as the basic form of knowledge formation in the human short-term memory. In this theory rules become the basic building blocks of human knowledge structures. This sense of rules gives rule-based systems an aura of legitimacy as models of intelligence. But we feel that one should be very cautious about making this connection, even though it seems a natural one in the light of the fact that expert systems are a branch of artificial intelligence. This interpretation is neither necessary nor sufficient for the role of rules in expert systems. It is not necessary because, while AI programs may advantageously model human thought at some level of abstraction, it is not obvious that it should do so at the level of knowledge formation in short-term memory, especially for capturing expert problem-solving performance. It is not sufficient because even if rules were the basic units of cognition, a number of further constructs are needed to account for their organization into higher level units such as concepts, and for their interaction with problem solving.

4.4. When Are Rule-Based Systems Appropriate?

The term "rule-based system" has come to mean an expert system which has a knowledge base of rules and a problem solver — such as a forward-chaining system or a backward-chaining system, or a production system controller such as OPS5 (Forgy & McDermott, 1977) — that uses the knowledge base to make inferences. It is conceptually important to keep in mind that the use of rules as a representation device does not necessarily force one to use the

rule-based system architecture mentioned above, i.e., rules can be used for expert system design in significantly different architectures. MDX (Chandrasekaran & Mittal, 1983), e.g., has some of its medical knowledge in the form of rules, but it is organized as a hierarchical collection of "specialists." We shall discuss this system later in the paper. The remarks that follow apply to the use of rules in the standard rule-based system architectures.

Because the problem solver (or, the inference engine, as it has come to be called) is itself free of knowledge, controlling the problem solving process often involves placing more or less complex rules for control purposes in the knowledge base itself. This is the "programming technology" sense of the use of rule based systems that was mentioned earlier. Normally, the rule-based architecture mentioned above works quite well when relatively little complex coupling between rules exists in solving problems, or the rules can be implicitly chunked into groups with little local interaction between rules in different chunks. R1 is an example where there is a virtual chunking of rules for subtasks, and the reasoning proceeds in a relatively direct and focused way. In general, however, when the global reasoning requirements of the task cannot be conceptualized as a series of linear local decisions, the rule-based systems of the simple architecture results in significant "focus" problems, i.e., since the problem solver does not have a notion of purposes at different levels of abstraction, there are often problems in maintaining coherent lines of thought. Focus needs maintenance of multiple layers of contexts, goals and plans. We shall discuss later how alternative architectures may be conceived for better focus in problem solving. These

architectures will begin to erase the separation of knowledge base from the inference machinery.

There is one pragmatic aspect of rule-based approaches to expert system design that is important. Since only certain kinds of expertise have natural expression as rules, a rule-based approach may encourage a false feeling of security in capturing expertise. The human expert often may not bring up expertise hard to express in that form. The 20%/80% phenomenon is worth recalling here: i.e., as the most rule-like pieces of knowledge get encoded, there may be an acceptable initial performance, but as more knowledge that is not naturally rule-like is being acquired, the size of the rule base may grow very rapidly.

5. ON LOGIC AND FRAMES AS REPRESENTATION FORMALISMS

The architecture of systems using some sort of a logical formalism for knowledge representation is generally similar to that of the rule-based systems discussed earlier. That is, there are a knowledge base of formulas in some logical formalism, and an inference machinery that uses the formulas to make further inferences. But since there are several forms of logic with well-understood semantics — unlike, say, production rules — logic enjoys a status as theoretically more rigorous for knowledge representation purposes. In practical terms, however, the existence of rigorous semantics is not always helpful, since the semantics are often not at the right level of abstraction; e.g., it is often difficult to incorporate context-dependent inference strategies in logic-based systems. If one were to model, say, reasoning in arithmetic, one could represent domain knowledge in the form of axioms, and use a variety of inference machineries to derive new theorems. However, the

computational complexity of such systems tends to be impractical even for relatively simple axiomatic systems. Even in such domains where in theory powerful axiom systems exist, capturing the effectiveness of human reasoning is an open research area. On the other hand, in tasks where the inference chains are relatively shallow, i.e, the discovery of the solution does not involve search in a large space, the logic representation may be more practical. Analogous to our remarks regarding the appropriateness of rule-based systems when there is an implicit structure to the task that can be mapped into chunking of rules, in logic-based systems also it is possible to create a similar subtasking structure that can keep the complexity of inference low.

To our mind, both the logic-based architectures and the rule-based architectures have surprising similarities in some dimensions; in both of them, the architecture separates a knowledge base from mechanisms that use the knowledge. In both cases, this results in an increasing need to place in the knowledge base more and more control-type knowledge — the representations increasingly become programming technologies rather than perspicuous encodings of problem solving activity. Because of an inability at that level to specify complex structures such as contexts and goal hierarchies, the approaches are subject to problems of focus in reasoning.

Control of problem solving requires, in our opinion, organizing knowledge into chunks, and invoking portions of the knowledge structures and operating on them in a flexible, context-dependent manner. The knowledge representation approach in AI that first emphasized organization in the form of structured units was that of frames (Minsky, 1975). Frames are especially useful in organizing a problem solver's "what" knowledge— knowledge about objects,

e.g. — in such a way that efficiency in storage as well as inference can be maintained. Assuming some familiarity on the part of the reader with the frame concept, we will briefly mention three aspects of frames that contribute to this efficiency. Basically, a frame of a concept being a structured stereotype, much of the knowledge can be stored as "defaults," and only the information corresponding to differences from the default value needs to be explicitly stored. For example, the default value of the number of walls for a room is 4, the knowledge of the system about a particular room does not have to explicitly store this, unless it is an exception, e.g., it is a 5-walled room.

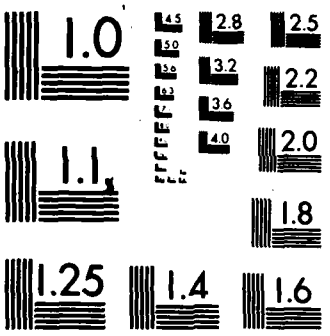
A second benefit of organizing one's knowledge of objects in the form of frame structures is that one can create frame hierarchies, and let much of the knowledge about particular objects be inherited from information stored at the class level. This again makes for great economy of storage. For instance, the "purpose" slots of a bedroom and a living room may be different, but the parts that are common to them, e.g., typically all rooms have four walls, a ceiling etc., can be stored at the level of the "room" frame, and inherited as needed at the "bedroom" or "living room" level.

A third mechanism that makes frame structures very useful in expert systems is the possibility of embedding procedures in the frames so that certain inferences can be performed as appropriate for the conceptual context. This, as can be seen, is a move away from the rule-based system architecture, where the inference mechanisms were divorced from the knowledge base.

Because of these three properties, frame systems are very useful for capturing one broad class of problem solving activity, viz., one where the basic task can be formulated as one of <u>making inferences about objects by</u>

AD-A137 828 DISTRIBUTED KNOWLEDGE BASE SYSTEMS FOR DIAGNOSIS AND INFORMATION RETRIEVA..(U) OHIO STATE UNIV COLUMBUS DEPT OF COMPUTER AND INFORMATION SCI.. B CHANDRASEKARAN 2/2

UNCLASSIFIED NOV 83 AFOSR-TR-84-0039 AFOSR-82-0255 F/G 9/4 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

using one's knowledge of related objects elsewhere in the structure. Thus a whole class of programming styles called "object-oriented" programming has arisen which has conceptual kinship with the notion of frames.

6. ORGANIZATION OF KNOWLEDGE

Even though many knowledge-base systems follow the popular architecture of a knowledge base (generally of rules, but many times of other forms: networks in Prospector, frames in Pip) and an inference engine,[3] often they have further internal architectures, but which are implicit in the sense that they are accomplished, as mentioned in an earlier section, by using programming techniques (again, typically using rules to specify conditions for transfer to the module) to achieve some degree of modularization of the knowledge into groups. Before we discuss this, however, some account of the need for this sort of organization is called for.

When the number of domain rules in the knowledge base is large, typically several rules will "fire," i.e., their left hand sides will match the state of the data. Since the inference machinery (because in the standard architecture it is deliberately kept domain-independent) does not have the domain knowledge to choose among them, either some sort of syntactic conflict-resolution mechanisms need to be used (such as the technique of R1 which chooses that rule whose conditions strictly subsume those of a contending rule), or all of them will need to be tried. The latter option has the potential for combinatorial explosion. Most systems attempt to cope with this problem by

---

[3] We will argue in Sec. 7 against this separation but for our immediate purposes that is not important.

creating "contexts," which help specify a small set of rules in the base as candidates to be considered for matching. For example, each subtask in R1 creates a context and only rules relevant to that subtask are considered for matching. This technique essentially decomposes the rule base (except for the rules which effectuate the transfer from module to module) into a number of virtual modules, each for a subtask. Prospector (which is a geological consultation system, (Duda et al, 1979)), while not a rule-based system explicitly,[4] also organizes its knowledge in the form of "models," each model corresponding to a classification hypothesis about the geological make-up. In a sense, each model is a "specialist" in that hypothesis. Metarules (Davis, 1976) have been proposed as a special class of rules to embody "control knowledge." These also play the role of decomposing the knowledge base into portions that are relevant for classes of situations. Without some such attempt at organization, the problem solving process will be generally be very unfocused, and serious control problems will arise.

All the above organizational devices were implicit, and are subject to the constraints of the rule formalism on the one hand, and the uniformity of the inference procedure on the other. The uniformity of the inference machinery makes it difficult to arrange for different subtasks during reasoning to exploit different ways of going about using the knowledge. Again it is worth emphasizing that the issue is not one of the computational sufficiency of the rule mechanism, but one of naturalness and conceptual adequacy.

---

[4] but the content of the network representation can be translated into rule forms in a straightforward way

Our own work (e.g, (Chandrasekaran, 1983), which gives an overview of our activity) has been directed toward the development of a theoretical basis of knowledge organization for expert problem solving. We will outline some aspects of this theory in the next section.

7. CONCEPTS AND PROBLEM SOLVING TYPES AS ORGANIZATIONAL CONSTRUCTS

To restate a point from the last section: Even the implicit modularization of the knowledge base due to various context-setting mechanisms is not always sufficient when the task domain consists of subtasks which might differ from each other in the nature of the problem solving, i.e., the use of knowledge that is required for that subtask. The work to be discussed has been directed toward elaborating a framework in which different generic types of problem solving can be related to the types of knowledge organizations required for them.

7.1. Generic Tasks

The theory proposes that there are well-defined generic tasks each of which calls for a certain organizational and problem solving structure. We have identified several such generic tasks from our work in the domains of medicine and reasoning about engineered systems.

The classificatory task that is at the core of medical diagnosis, i.e., the task which classifies a complex case description as a node in the disease hierarchy is an example of such a generic task. It is generic because it is a component of many real world problem solving situations. For example, a tax-advising expert program might go through a stage of classifying the user as a particular type of taxpayer before invoking strategies that may be

appropriate for that class of taxpayer. We have already discussed in Sec. 3 simpler versions of this task. The problem solving for this task, as implemented in our medical diagnosis system, MDX, will be considered in Sec. 7. 1. 1.

Another example of a generic task is what we call a WWHI-type (for "What Will Happen If") reasoning which attempts to derive the consequences of an action that might be taken on a complex system. Such a task is useful as a subtask in an expert system that trouble-shoots and repairs a complex system where it may be useful to reason out the consequences of a proposed corrective action.

A third type is a form of knowledge-directed associative memory that helps retrieve information by reasoning about other related information; we have used this type of problem solving in an intelligent data base system, PATREC (Mittal and Chandrasekaran, 1980). A fourth type is a form of plan synthesis, which we are using to build an expert system for mechanical design (Brown and Chandrasekaran, 1983). It is clear that there are many more such generic tasks, and it is part of our research program to identify more of them.

An important consequence of identifying such tasks is that it gives us a framework to characterize the capabilities of expert systems. If a real world task can be decomposed into a number of generic tasks for each of which we know how to build a reasoning system, then there will be a basis for concluding that the task domain can be successfully tackled by an expert system.

In the next two subsections, we will discuss in greater detail, but still in schematic terms, how knowledge can be organized and problem solving can be accomplished for two of the above generic tasks. Cited references can be

consulted both for more details on these tasks, as well as for information on the other two problem solving types that we omit here due to space limitations.

7.1.1. The Classificatory Task

As mentioned earlier, the task is the identification of a case description with a specific node in a pre-determined diagnostic hierarchy. For the purpose of current discussion let us assume that all the data that can be obtained are already there, i.e., the additional problem of launching exploratory procedures such as ordering new tests etc. does not exist. The following brief account is a summary of the more detailed account given in Gomez and Chandrasekaran (1981) of diagnostic problem-solving.

Let us imagine that corresponding to each node of the classification hierarchy alluded to earlier we identify a "concept." The total diagnostic knowledge is then distributed through the conceptual nodes of the hierarchy in a specific manner to be discussed shortly. The problem-solving for this task will be performed top down, i.e., the top-most concept will first get control of the case, then control will pass to an appropriate successor concept, and so on. In the medical example, a fragment of such a hierarchy might be as shown in Figure 1.

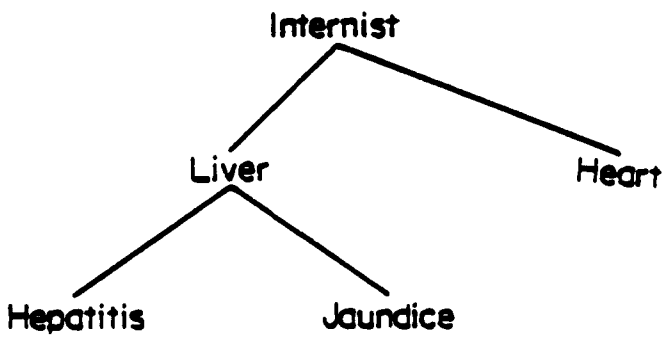


Figure 1. Fragment of a classificatory hierarchy

More general classificatory concepts are higher in the structure, while more particular ones are lower in the hierarchy. It is as if INTERNIST first establishes that there is in fact a disease, then LIVER establishes that the case at hand is a liver disease, while say HEART etc. reject the case as being not in their domain. After this level, JAUNDICE may establish itself and so on.

The problem-solving that goes on in such a structure is distributed. The problem-solving regime that is implicit in the structure can be characterized as an "establish-refine" type. That is, each concept first tries to establish or reject itself. If it succeeds in establishing itself, the refinement process consists of seeing which of its successors can establish itself. The way in which each concept (or "specialist") attempts to do the establish-refine reasoning may vary from domain to domain. In medicine it may often be accomplished by using knowledge in the form of a collection of rules, some of which look for evidence for the hypothesis, some for counter evidence, and others which carry information about how to combine them for a final conclusion. In reasoning about electrical circuits on the other hand it may be more appropriate to represent the establish-refine activity in the form of

functional knowledge about specific modules. (That is, performance of a generic task may require solution of some problem of a different type as a subtask.)

In our medical diagnosis system MDX, each of the concepts in the classification hierarchy has "how-to" knowledge in it in the form of a collection of diagnostic rules. These rules are of the form: <symptoms> ——> <concept in hierarchy>, e.g., "If high SGOT, add n units of evidence in favor of cholestasis." Because of the fact that when a concept rules itself out from relevance to a case, all its successors also get ruled out, large portions of the diagnostic knowledge structure never get exercised. On the other hand, when a concept is properly invoked, a small, highly relevant set of rules comes into play.

Each concept, as mentioned, has several clusters of rules: confirmatory rules, exclusionary rules, and perhaps some recommendation rules. The evidence for confirmation and exclusion can be suitably weighted and combined to arrive at a conclusion to establish, reject or suspend it. The last mentioned situation may arise if there is not sufficient data to make a decision. Recommendation rules are further optimization devices to reduce the work of the subconcepts. Further discussion of this type of rules is not necessary for our current purpose.

The concepts in the hierarchy are clearly not a static collection of knowledge. They are active in problem-solving. They also have knowledge only about establishing or rejecting the relevance of that conceptual entity. Thus they may be termed "specialists," in particular, "diagnostic specialists." The entire collection of specialists engages in distributed problem-solving.

The above account of diagnostic problem-solving is quite incomplete. We

have not indicated how multiple diseases can be handled within the framework above, in particular when a patient has a disease secondary to another disease. Gomez has developed a theory of diagnostic problem-solving which enables the specialists in the diagnostic hierarchy to communicate the results of their analysis to each other by means of a blackboard and how the problem-solving by different specialists can be coordinated. Similarly, how the specialists combine the uncertainties of medical data and diagnostic knowledge to arrive at a relatively robust conclusion about establishing or rejecting a concept is an important issue, for a discussion of which we refer the reader to Chandrasekaran, Mittal and Smith (1982).

The points to notice here are the following. The inference engine is tuned for the classificatory task, and the control transfer from specialist to specialist is implicit in the hierarchical conceptual structure itself. One could view the inference machinery as "embedded" in each of the concepts directly, thus giving the sense that the concepts are "specialists."

7.1.2. What-Will-Happen-If (WWHI) or Consequence Finding

Examples of this type of reasoning are: "What will happen if valve A is closed in this power plant when the boiler is under high pressure?"; "What will happen if drug A is administered when both hepatitis and arthritis are known to be present?" Questions such as this can be surprisingly complex to answer since formally it involves tracing a path in a potentially large state space. Of course what makes it possible in practice to trace this path is domain knowledge which constrains the possibilities in an efficient way.

The problem-solving involved, and correspondingly the use of knowledge in this process, are different from that of diagnosis. For one thing, many of

the pieces of knowledge for the two tasks are completely different. For example, consider answering the question in the automobile mechanic's domain: "What will happen if the engine gets hot?" Looking at all the diagnostic rules of the form, "hot engine ——> <malfunction>" will not be adequate, since <malfunction> in the above rules is the cause of the hot engine, while the consequence finding process looks for the effects of the hot engine. Formally, if we regard the underlying knowledge as a network connected by cause-effect links, where from each node multiple cause links as well as effect links emanate, we see that the search processes are different in the two instances of diagnosis and consequence-finding. The diagnostic concepts that typically help to provide focus and constrain search in the pursuit of correct causes will thus be different from the WWHI concepts needed for the pursuit of correct effects.

The embedded problem-solving is also correspondingly different. We propose that the appropriate language in which to express the consequence-finding rules is in terms of state-changes. To elaborate:

- 1. WWHI-condition is first understood as a state change in a subsystem.
- 2. Rules are available which have the form "<state change in subsystem> will result in <state change in subsystem>". Just as in the case of the diagnosis problem, there are thousands of rules in the case of any nontrivial domain. Again, following the diagnostic paradigm we have already set, we propose that these rules be associated with conceptual specialists. Thus typically all the state change rules whose left hand side deals with a subsystem will be aggregated in the specialist for that subsystem, and the right

hand side of those rules will refer to the state changes of the immediately affected systems.

Again we propose that typically the specialists be organized hierarchically, so that a subsystem specialist, given a state change to it, determines by knowledge-based reasoning the state changes of the immediately larger system of which it is a part and calls that specialist with the information determined by it. This process will be repeated until the state change(s) for the overall system, i.e., at the most general relevant level of abstraction, are determined. This form of organization of the rules should provide a great deal of focus to the reasoning process.

An Illustrative Example.

Consider the question, in the domain of automobile mechanics, "WWHI there is a leak in the radiator when the engine is running?" We suggest the specialists are to be organized as in Figure 2 :

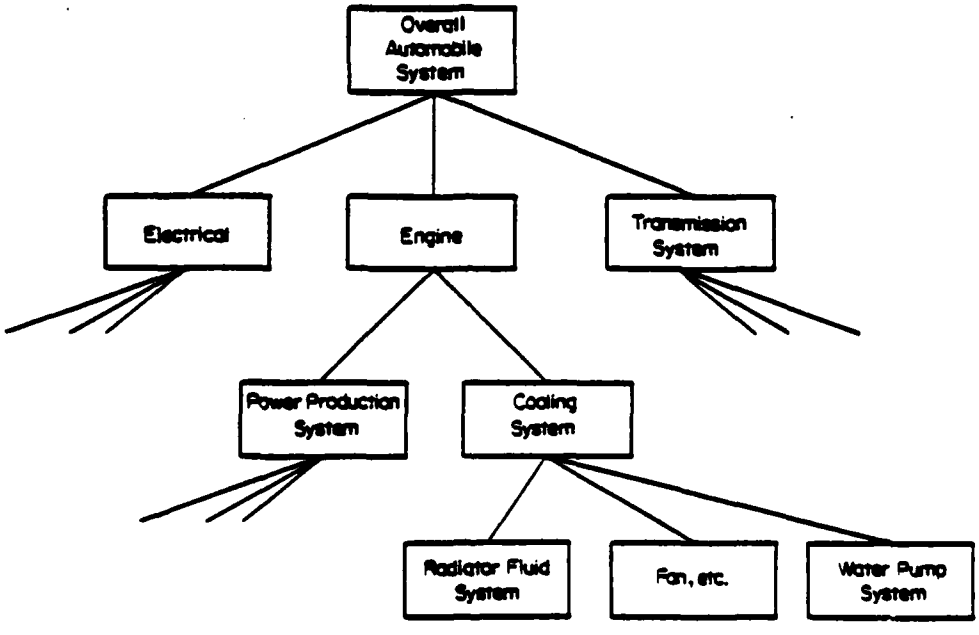


Figure 2. Example of WWHI concept hierarchy

The internal states that the radiator fluid subsystem might recognize may be partially listed as follows: {leaks/no leaks, rust build-up, total amount of water,... }; similarly, the fan subsystem specialist might recognize states {bent/straight fan blades, loose/tight/disconnected fan belt,...}. The cooling system subsystem itself need not recognize states to this degree of detail; being a specialist at a somewhat higher level of abstraction it will recognize states such as {fluid flow rate, cooling-air flow rate...etc.}. Let us say that the radiator fluid specialist has, among others, the following rules. The rules are typically of the form:

<internal state change> ——> <supersystem state change>

leak in the radiator ——> reduced fluid flow-rate

high rust in the pipes ——> reduced fluid flow-rate

no antifreeze in the water and very cold weather ——> zero fluid flow etc.

The cooling system specialist might have rules of the form:

low fluid-flow rate and engine running ——> engine state hot

low air-flow rate and engine running ——> engine state hot

Again note that the internal state recognition is at the appropriate level of abstraction, and the conclusions refer to state changes of its parent system.

It should be fairly clear how such a system might be able to respond to the query about radiator leak. Again a blackboard for this task would make it possible to take into account subsystem interaction.

Unlike the structures for the diagnostic and data retrieval tasks, we have not yet implemented a system performing the WWHI-task. While we cannot speak with assurance about the adequacy of the proposed solution, we feel that it is of a piece with the other systems in pointing to the same set of morals: embedding still another type of problem-solving in a knowledge structure, which consists of cooperating specialists of the same problem-solving type.

7.2. Discussion

Since each of the generic tasks involves a problem-solving behavior which is unique to that task, the standard architecture of a knowledge base and a general purpose inference machinery is not applicable here. There is a closer intertwining of knowledge structures and corresponding inference methods. At the implementation level, one can view the system as being decomposed into a collection of pairs of the form (<knowledge structure, inference method>), indexed by the generic tasks, e.g., <diagnostic structure, establish-refine>. However it is conceptually more appropriate to view each of the specialists as having the inference machinery "embedded" in them. This interpretation gives

the term "specialists" an added degree of aptness.

In Sec. 2 we mentioned that the first generation of expert systems emphasized the power of knowledge itself over that of the problem solving method. In the current section we have attempted to restore the balance, by showing how a variety of problem-solving types in conjunction with appropriate organizations of knowledge can solve a greater variety of problems.

8. EXPERT SYSTEMS THAT UNDERSTAND

We have outlined an evolution of expert systems from collection of rules to cooperative problem solving by a community of specialists in different kinds of problem solving. The knowledge that is in each of the specialists, e.g., the diagnostic rules in the classificatory specialists or the state abstraction rules in the WWHI specialists in the previous section, is itself "compiled." This knowledge is obtained from human experts who either learnt that knowledge originally in that compiled form, formed it as a result of experience, or <u>derived</u> it from a deeper model of the domain. In Chandrasekaran and Mittal (1982) we argue that in principle, given any deep model of the domain, one can compile an MDX-like diagnostic system which is as powerful as the deeper model, but more efficient than it, for the diagnostic problem. However, in practice, the compiled structures are likely to remain <u>incomplete</u> for various reasons, and it would be very attractive to endow expert systems with deeper understanding of their domains to protect themselves against incompleteness.

Attempts to give expert systems some ability to do deeper level reasoning have typically taken the direction of giving the system a mechanism to reason at more or less levels of detail by using a prestored knowledge base of causal

associations. In CASNET (Weiss, et al., 1978) an attempt is made to trace out the most likely causal sequence given some likely intermediate states for which there is evidence and some states which can be assumed not to have occurred. A knowledge base of possible causal connections between states with associated likelihood information is used to fit a most likely path that goes through the states for which there is evidence and avoids the unlikely states. ABEL (Patil, 1981) uses causal association information at different levels of detail. There may be a piece of knowledge at the top level of the form, "A causes B," but at a more detailed level, there may be several different ways in which A might be able to cause B. The system works at these different levels of detail to pin the causal connection down to the degree that is required.

It is important to note that these systems do not use "causal models" as much as they use a storehouse of compiled causal associations. In a sense all diagnostic programs use "causal models." Much of the diagnostic knowledge in MYCIN or MDX is causal, i.e., saying "symptom A gives so much evidence for disease B," is, in content, the same as "B causes A with so much likelihood." The difference between them and the programs above is what is done with the causal knowledge.

In our view a truly deep model should have the power to derive the causal connections between states. The work of Kuipers (1982), deKleer and Brown (1981) and Chandrasekaran and Moorthy (forthcoming) are relevant here. Kuipers proposes methods by which causal behavior may be derived from a knowledge of the structure of some system. The latter two references seek to model functional understanding of devices.

The scope of the current paper does not allow a detailed description of

how the functional representations work and how they are related to diagnostic reasoning. Here we content ourselves with an intuitive account of our approach.

We model understanding of a device as the creation of a knowledge structure which is hierarchically organized in terms of functions and subfunctions. How the salient behavior (generally stated in qualitative terms) and the physical structure of the relevant portions of the device play a role in achieving a function by means of the subfunctions is part of such a description.

We believe that such a structure can be used to generate the causal knowledge needed for diagnostic reasoning. The function/subfunction relationship can be used to generate diagnostic hypotheses. If function A is affected, each of its subfunctions can be considered as a possible source of failure. Similarly the symptomatic knowledge that is needed for establishing or rejecting these as possibilities can be derived from the behavioral and physical structure constraints that enable subfunctions to be achieved; "if behavior B is necessary to accomplish subfunction A', look for evidence or lack of B in the behavior of the device," e.g., would be a useful way to generate some of that knowledge. The attempt to give systems a degree of understanding based on functional models is still very experimental, and practical expert systems based on this approach are still some time away. But we feel that this sort of systems is the next step in the evolution of expert systems towards a greater degree of understanding.

9. DISCUSSION

The reader may have gathered that there is no simple method of determining which tasks are likely to be successfully handled by an expert system. We have attempted to give the reader some analytical tools by which such a decision may be made a little easier. In any case, the following guidelines arise from our experience in designing a number of expert systems.

1. When the total amount of knowledge is relatively small (a few hundred rules), the exact technique used is not very important. A wide variety of techniques will all give similar qualitative performance.
2. A large fraction of expert systems that have reached some degree of exposure (Prospector, Mycin, MDX, CASNET, Internist) deals with some form of the classification problem. If the core task in an application domain is classificatory, chances are very good that an expert system approach will be successful. Problems of synthesis, such as design, are in general harder, but some simpler versions of the design problem, such as the task domain of R1, have been successfully attacked.
3. Another type of expert system that is likely to be of practical applicability is one that helps the user access knowledge in a complex knowledge base. This type of expert behavior does not require the full problem solving capabilities of the expert. Our work on intelligent data bases (Mittal and Chandrasekaran, 1980) offers some techniques of potential relevance here.
4. If a real-world task can be decomposed into a number of generic

tasks for which expert system solutions are available, then the prospect of a successful expert system increases significantly.

5. Important research is going on in common sense qualitative reasoning involving space and time; these advances will give the next generation of expert systems more power and flexibility.
6. Reasoning of experts is in general varied and broad-ranging. We have only begun to understand some forms of such reasoning. Honesty compels us to admit that it is not a simple matter to capture all forms of expertise and incorporate them in the form of computer programs, even though, in the enthusiasm surrounding this promising field, careful distinctions and qualifications often do not get made. On the positive side is the solid body of accomplishment: the field has managed to capture a number of useful forms of expertise.

We have not discussed in this paper a number of issues such as user interfaces, explanation facilities, and knowledge acquisition problems. They are obviously of great practical importance, but it has always seemed to us that the issues of knowledge organization and problem solving will continue to occupy the center stage in this area, since these problems are by no means solved.

References

Brown, D.C. & Chandrasekaran, B., "Expert systems for mechanical design," Proc. IEEE Trends & Applications in AI, Washington, D.C., May, 1983, pp. 173-180.

Chandrasekaran, B., "Toward a taxonomy of problem solving," AI Magazine, Vol. IV, No. 1, Winter/Spring 1983, pp. 9-17.

Chandrasekaran, B. & Mittal, S., "Deep versus compiled knowledge

approaches to diagnostic problem-solving," Proc. of American Association of Artificial Intelligence Conference, 1982, Pittsburgh, PA, pp. 349-354. To appear in Intnl. J. of Man Machine Studies.

Chandrasekaran, B. & Mittal, S., "Conceptual representation of medical knowledge," in Advances in Computers, M. Yovits (ed.), Academic Press, New York, 1983, Vol. 22, pp. 217-293.

Chandrasekaran, B., Mittal, S. & J. W. Smith, M.D., "Reasoning with uncertain knowledge: the MDX approach," Proc. 1st Ann. Joint Conf. of the American Medical Informatics Association, May, 1982, pp. 335-339.

Chandrasekaran, B. & Sembugamoorthy, V., "Functional models and diagnostic expertise formation," in preparation.

Davis, R., "Applications of metalevel knowledge to the construction, maintenance, and use of large knowledge bases," Ph. D. Dissertation, Stanford University, 1976.

Davis, R., "Expert systems: Where are we? And where do we go from here?", AI Magazine, Vol. 3, No. 2, Spring 1983, pp. 3-22.

deKleer, J. & Brown, J. S., "Mental models of physical mechanisms and their acquisition," Cognitive Skills and Their Acquisition, Anderson (ed.), Erlbaum Associates, Hillsdale, N. J., 1981.

Duda, R. O., Gaschnig, J., Hart, P. E., "Model design in the prospector consultant system for mineral exploration," in Expert Systems in the Micro-Electronic Age, D. Michie (ed.), Edinburgh University Press, 1979, pp. 153-167.

Forgy, C. & McDermott, J., "OPS: A domain-independent production system," Proc. of the Fifth Intnl. Joint Conf. on AI, 1977, pp. 933-939.

Gomez, F. & Chandrasekaran, B., "Knowledge organization and distribution

for medical diagnosis," IEEE Trans. Systems, Man, and Cybernetics, Vol. SMC-11, No. 1, January 1981, pp.34-42.

Kuipers, B., "Commonsense reasoning about causality: Deriving behavior from structure," Tufts University, Working Papers in Cognitive Science, No. 18, May 1982.

Marr, D., "Early processing of visual information," Phil. Trans. of the Royal Society of London, Vol. 275, Series B, 1976, pp. 483-524.

McDermott, J., "R1: A rule-based configurer of computer systems," Artificial Intelligence, Vol. 19, No. 1, 1982, pp. 39-88.

Minsky, M. "A framework for representing knowledge," The Psychology of Computer Vision, P. H. Winston (ed.), McGraw Hill, New York, 1975.

Minsky, M. & Papert, S., Perceptrons: an Introduction to Computational Geometry, MIT Press, Cambridge, Mass, 1969.

Mittal, S. & Chandrasekaran, B., "Conceptual representation of patient data bases," J. Medical Systems, Vol. 4, No. 2, 1980, pp. 169-185.

Newell, A., "Production systems: Models of control structure," in Visual Information Processing, W. Chase (ed.), Academic Press, New York, 1983, pp. 463-526.

Newell, A. & Simon, H.A., Human Problem Solving, Prentice-Hall, Englewood Cliffs, N.J., 1972.

Nilsson, N.J., Problem Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

Patil, R. S. "Causal representation of patient illness for electrolyte and acid-base diagnosis," Ph. D. Dissertation, TR-267, MIT Lab for Computer Science, Cambridge, Mass., October, 1981.

Pauker, S. G., Gorry, G. A., Kassirer, J. P. & Schwartz, W. B., "Towards

the simulation of clinical cognition: Taking a present illness by computer," The American Journal of Medicine, Vol. 60, 1976, pp. 981-995.

Samuel, A., "Some studies in machine learning using the game of Checkers, " IBM J. of Research & Development, 1967, pp. 601-617.

Stefik, M., Aikins, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., Sacerdoti, E., "The organization of expert systems: a prescriptive tutorial," Xerox PARC, Tech. report, 1982.

Szolovits, P., & Pauker, S. G., "Categorical and probabilistic reasoning in medical diagnosis," Artificial Intelligence, Vol. 11, 1978, pp. 115-144.

Weiss, S. M., Kulikowski, C. A., Amarel, S., & Safir, A., "A model-based method for computer-aided medical decision-making," Artificial Intelligence, Vol. 11, 1978, pp. 145-172.

END

FILMED

3-84

DTIC